Honeywell

CONQUER: Co-ordination and Optimization of Quality-of-Service End-to-end Resources for Adaptive Information Flows

Final Report

Allalaghatta Pavan (PI) Vipin Gopal Nigel Birch Sejun Song Raja Harinath Zhi-Li Zhang	David Castanon Yingtao Zuo	Marc Zev
Honeywell Technology Center 3660 Technology Drive Minneapolis, MN 55418	Boston University MDSP Lab., Electrical and Computer Eng. Dept. Boston, Massachusetts 02215	ISX Corporation 4353 Park Terrace Dr West Lake Village CA 91361

Sponsored by
Defense Advanced Research Projects Agency
Contract No. DABT63-99-C-0005

May 31, 2000

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 2000	Final Technical Report, January 1999 to April 2000

4. TITLE AND SUBTITLE

CONQUER: Co-ordination and Optimization of Quality-of-Service End-to-end Resources for Adaptive Information Flows

5. FUNDING NUMBERS

C - DABT63-99-C-0005

6. AUTHOR(S)

Allalaghatta Pavan, Vipin Gopal, Nigel Birch, Sejun Song, Raja Harinath, Zhi-Li Zhang (Honeywell); Marc Zev (ISX); David Castanon, Yingtao Zuo (Boston Univ.)

7 PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55418

ISX Corporation
4353 Park Terrace Dr
West Lake Village, CA 91361

Boston University
MDSP Lab., Electrical and Computer Eng. Dept.
Boston, MA 02215

8. PERFORMING ORGANIZATION REPORT NUMBER

None

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTAL NOTES

12A. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Highly dynamic information environments that support military applications should be able to satisfy increasingly stringent demands on end-to-end QoS requirements. As part of the DARPA AICE (Agile Information Control Environment) program, we have developed a middleware framework for admission control and resource allocation, for maximization of the end-to-end QoS for communication channels. The framework includes a novel architecture for the control environment, and rich models for flow characterization and resource constraints. Algorithms based on various optimization criteria are developed for resource allocation that maximize the overall mission objectives. In this report, the overall architecture is described along with various resource allocation algorithms. Different test cases demonstrate the utility of such optimal resource allocation mechanisms for the success of the overall mission.

14. SUBJECT TERMS

resource allocation; adaptive flows; mission planning; optimization; end-to-end QoS

15. NUMBER OF PAGES

54

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

Section 1 Overview of CONQUER

1.1 Introduction

Agile Information Control Environment (AICE) provides capabilities required to observe, understand, predict, manage, and, especially, control information flows in support of military operations. AICE technologies enable information infrastructure management to be mission driven (i.e., infrastructure configuration is dynamically configured so that information is exchanged in a manner commensurate with mission objectives). Real-time information flows, supporting mission- or life-critical applications within a shared communication environment, are the primary motivators for the AICE development effort. AICE enables dynamic information control for the war-fighter on the battlefield through the use of quality of service (QoS) measures.

This report first describes the overall AICE architecture, and focuses on the design, implementation and performance of the Adaptive Information Control component developed by Honeywell, Boston University and ISX Corporation.

1.2 AICE System Architecture

The AICE system is composed of three interacting functional components: Information Policy Management (IPM) [Funk00], Adaptive Information Control (AIC), and MetaNet [Fal00]. IPM uses multiple commanders' policies regarding network resource allocation, resolves any differences among them, and computes the importance of individual requests in response to queries from the AIC component. The AIC component tracks the usage of current network resources and decides whether or not to grant the request, and if granted, the level of the resource to allocate. MetaNet interacts with the underlying networks to set up the channel and provides a uniform interface to a set of underlying heterogeneous networks.

All components interact with each other via well-defined interfaces expressed in the Interface Definition Language (IDL). The AICE components and their IDL interfaces were jointly developed by the AICE program participants [TRW99].

Figure 1 illustrates the basic interactions associated with channel request submission. External clients (e.g., military personnel) make a request for a channel, specifying the time and duration of the channel and associated channel QoS parameters, such as bandwidth and the 'utility' of receiving a specific bandwidth. AIC receives the request and determines both the "importance" of the request, and the available resources from IPM and MetaNet respectively. AIC determines if the channel can be allocated or not and informs the external client of the result.

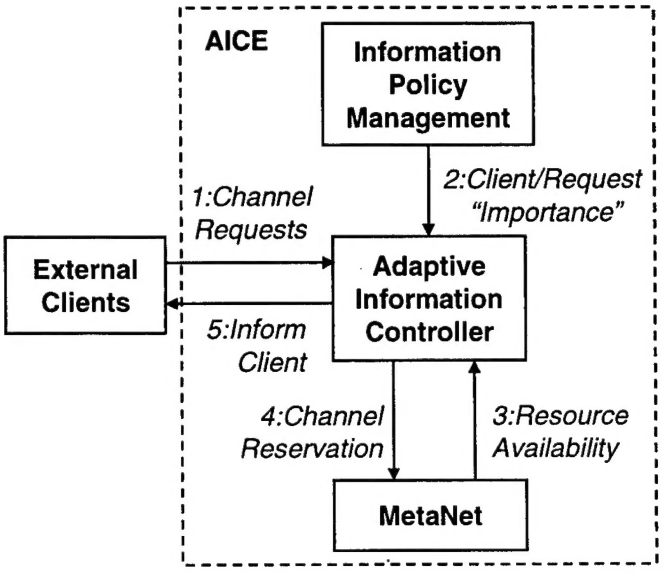


Figure 1 - AICE Functional Components

Each AICE component is described in more detail the following sections.

1.2.1 IPM

Information Policy Management (IPM) provides an interface to commanders spanning all echelons to control the allocation of resources used to exchange information across the battle-space in accordance with mission objectives. In IPM, disparate policy inputs from commanders are formulated to be a consistent and covering policy. In this approach, a policy represents the commander's desired allocation of communications resources during the execution of a mission; the policy takes the form of a set of general and specific statements about the priorities, constraints and objectives for information flow. IPM assesses the "importance" of a channel request based on its internal policy. The importance is quantified and used by AIC to prioritize channel allocations.

1.2.2 MetaNet

MetaNet serves as a uniform AIC entry point and resource assignment mechanism for channels that span heterogeneous networks. MetaNet provides the concept of abstract channels on top of the physical resources via Network Adapters. MetaNet 'wraps' the actual networks that support connectivity between host machines. MetaNet defines a Point of Presence (PoP) as the point of attachment of one or more host machines. Via MetaNet PoPs, MetaNet hides the technology specific details of the underlying networks and allows QoS provisioning across military and commercial networks. One or more host machines are called a "hostgroup". Each hostgroup is uniquely named and made known to MetaNet. MetaNet defines a single *logical* path between each host group pair as MetaNet Links. The physical path that underlies a MetaNet Link can change over time and thus the available network capacity of a MetaNet Link may change over time. In addition, two or more logical MetaNet Links may be 'dependent' on each other due to

their mutual reliance on one or more *physical* links. MetaNet provides priority assignment to channels, so that the existing channels can be preempted by ones of higher priority.

1.2.3 AIC

AIC performs two central functions, policy application and resource allocation. As stated earlier, an external client submits a channel request with a QoS parameter in the form of a discrete bandwidth/utility curve. The Channel Request Processor (CRP) obtains the importance of the request from IPM, and scales the requested utility to form a "conditioned" channel request, which is submitted to the Resource Allocator (RA). Currently our implementation does not condition the Utility function. The RA then obtains the latest resource availability information from MetaNet and attempts to allocate the new requested channel. This may result in either a Reject, or Accept accompanied by the set of existing channels that must be preempted or modified to guarantee the acceptance. The RA determines an allocation of network resources (e.g., bandwidth) in order to maximize the aggregate utility or 'value' of the allocated channels. If the channel was accepted, the CRP performs channel setup with MetaNet. The channel allocation and optimization algorithm will be described in the next section.

1.3 AIC Channel allocation Algorithm

We discuss here the primary algorithm developed by Honeywell Technology Center. Other algorithms developed are enclosed in the sections to follow. The discussion below will use the following terms and definitions:

- *Priority:* Every channel request is assigned a certain priority based on the importance of the request. In this paper, we will use integer values from 1 to 10. In the ensuing discussion, note that a lower numerical value for priority implies a higher priority (1 is the highest).

- *Time Intervals:* Variables such as network capacity are functions of time. Here, we abstract this information in terms of finite intervals of time, typically one hour. MetaNet currently reports its available Link capacity in terms of one hour time intervals, up to a maximum of 24 hours

- *Available Capacity:* The available capacity of a link in a particular time interval, is the capacity that is free to be allocated between the corresponding hostgroups, in that time interval. Note that this information is a function of time (interval). Available capacity for a link for a certain priority is available capacity assuming that all channels of lower priority are preempted.

- *Maximum Capacity:* The maximum capacity of a link between two hostgroups is the sum of capacities of all potential channels that can be set up between the hosts in these hostgroups. This information is, for practical purposes, quasi-static.

- *Link Dependencies:* Links L1 and L2 are said to be dependent if they share common resources for channel set-up. Link is an abstract concept – hence, it is not necessary that all channels set up over L1 should compete for resources with the others set up over L2,

however, they might be. Dependency is binary information – 1 if two links are dependent and 0, if not.

- *Utility function:* Each request quantifies its 'worth' with a Utility function. This captures the delivered utility to the channel request as a function of QoS. It could in principle be a continuous function, piecewise continuous, piecewise linear or even a simple set of discrete points. In this paper, we will consider a set of discrete points in 2-d space where the QoS attribute considered is bandwidth. Typically higher bandwidths are associated with higher utility.

- *Bottleneck Priority:* The highest priority level where there are not enough resources available to satisfy even the minimum requested QoS attribute. This is calculated for an individual time interval.

- *Run:* A consecutive series of time intervals that can support the requested resources for the requested duration.

The channel allocation problem will be posed as follows. Consider a request *i* of the form:

$r_i(L_i, p_i, \{u_{i,j}, q_{i,j}\}, t_{i,0}, t_{i,f,}, d)$.

Here, L_i is the corresponding MetaNet Link and p_i is the priority of the request. $\{u_{i,j}, q_{i,j}\}$ is the set of *j* discrete points of a utility function which maps each QoS parameter choices (bandwidth) to its utility. $t_{i,0}$, $t_{i,f}$ define the start and end time boundaries of the request, and d is the duration of the request channel.

The Admission Control and Resource Allocation (ACRA) problem is to compute:

- If channel *i* can be admitted.
- If it can be admitted:
  - When the requested channel will begin and end within $t_{i,0}$, $t_{i,f}$
  - The amount of resources that will be allocated to it, i.e., which point in the Utility Function was selected.
  - The set of allocated channels that need to be preempted/degraded for admitting *i*.

The following assumptions are made:

- A higher priority channel cannot be preempted if the requested resources can be freed up by preempting lower priority channels. This is due to the strictly priority based allocation policy followed.
- A channel of priority p may preempt one or more channels of the *same* priority based on delivered utility.
- We know the available link capacity per priority for the time intervals of the request.

- We know the Maximum MetaNet Link capacity.
- We know which MetaNet Links are dependent on each other.

Let T_i be the set of predefined time intervals overlapping and included in $(t_{i,0}, t_{i,f})$. Enough resources are available at a particular priority level p for a particular time interval t if:

$$\underset{t \in T_i}{Min}\{qa_p^t\} \geq \underset{j}{Min}\{q_{i,j}\} \qquad \textbf{(1)}$$

Here, qa_p^t is the available QoS parameter (e.g., bandwidth) in time interval t at priority p, and $\underset{j}{Min}\{q_{i,j}\}$ is the minimum value of the QoS parameter requested. For a given request, we use (1) to identify the bottleneck priority p_i^b for each time interval in T_i. This is compared with the priority of the request, with the following possible outcomes:

- $p_i^b > p_i$ - The time interval can support the request, i.e., there are enough resources to support request i by pre-empting some lower priority channels.
- $p_i^b = p_i$ - The time interval *may* be able to support the request if the utility of the request is sufficiently high. The utility optimizer will make the admission decision.
- $p_i^b < p_i$ - The time interval cannot support the request.

We then find 'runs' of successive time intervals in T_i for which $p_i^b > p_i$ or $p_i^b = p_i$ such that the duration d of the request can be accommodated. There may be many such candidate runs in T_i. In order to select which run to allocate, we apply the following test. For each candidate run we identify the most resource constrained time interval, i.e., that with the highest bottleneck priority p_i^b. Then we select the run with the lowest high p_i^b. This is the least bandwidth constrained, i.e., preempts the least number of lower priority channels.

Optimization occurs over the channels at the bottleneck priority. In the event that the priority of the submitted request is the bottleneck priority, the optimization determines the channel admission decision. In the event that the bottleneck priority is lower than the requested priority, the optimization selects those channels that deliver the most utility given the resources will have been reduced by the newly admitted channel.

We consider 'active' channels to be more valuable than 'pending' channels. While the priority of a channel cannot be changed, its utility can be modified to reflect the increased value of a channel based on its % completion. In this way, the optimizer will be less likely to preempt an active channel in order to admit a new channel.

An example formulation is follows. Let $E_{i,p}$={Set of all competing channels at priority p}. Let □ be the fraction of the time completed for a channel beyond which it is weighted higher. For example, if □ = 0.5, then channels which have completed half their duration are weighted higher. Let $M_{i,p}$={Set of all higher weighted channels}. Let M be defined such that M□>=1 for the channels in $M_{i,p}$. Set $L_{i,p}=E_{i,p}-M_{i,p}$, which includes the new request if the optimization problem is solved at the requested priority. The resulting optimization problem is:

$$
\begin{aligned}
&Max \sum_{i \in M_p} \sum_{j} M\alpha y_{i,j} u_{i,j} + \sum_{i \in L_p} \sum_{j} y_{i,j} u_{i,j} \\
&s.t. \quad \sum_{i} \sum_{j} y_{i,j} q_{i,j} \le q a_p \\
&\qquad \sum_{j} y_{i,j} \le 1 \qquad \forall i \\
&\qquad y_{i,j} \in \{0,1\}
\end{aligned}
\tag{2}
$$

This defines an objective function that maximizes the delivered utility, subject to the constraints that the available bandwidth is not exceeded, and only one choice is made from the bandwidth utility function points. Formulation (2) is an extension of the class of problems described in [Mart90, Lin98]

We have implemented this in the AIC RA.

1.4 AIC implementation

Honeywell has implemented AIC, and emulated versions of IPM, MetaNet and an external client in order to exercise and demonstrate AIC. The software is written using Microsoft Visual C++ 6.0 on Windows NT4.0. The inter-component CORBA interfaces are implemented using Iona Technologies Orbix V3.0.1 for Windows NT. We also use ILOG's CPLEX 6.5 linear programming problem solver.

The primary classes of the AIC component software are illustrated in Figure 2.

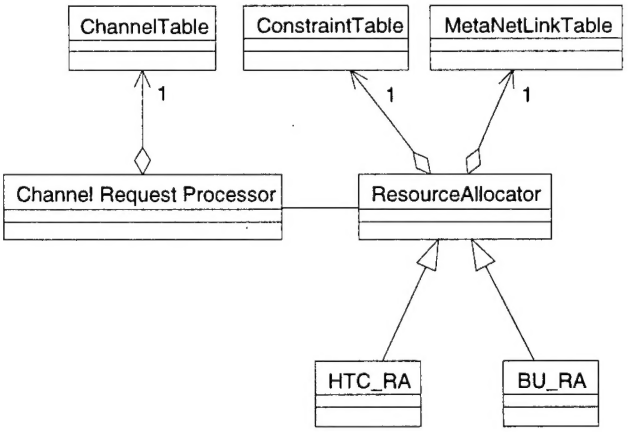


Figure 2 - AIC software architecture

AIC uses three data structures to manage its channel allocations: the channel table in the CRP, and the Constraint and MetaNet Link tables in the RA. The RA is implemented as a base class, which handles general table maintenance but no allocation algorithms. The allocation algorithms are implemented in RA subclasses. HTC and Boston University have each implemented a channel allocation algorithm as two independent subclasses.

The Channel Table is a dynamic list of entries, where each entry captures state information for a channel as it is submitted for admission, awaits activation, runs, and eventually completes (or is preempted). The MetaNet Link table contains information regarding link dependencies, and total link capacity information. Currently, this is a static table initialized once from MetaNet. This table must eventually become dynamic since Link capacities and dependencies may change over time. The Constraint table is a 3-dimensional array of MetaNet Link capacity data organized by time interval and priority level. This table is updated with Link capacity information from MetaNet prior to each allocation decision.

The following call sequence describes the channel request submission and allocation procedure implemented by the AIC component.

1. GetEndToEndConfigurationConstraints() called by AIC to initialize the RA's MetaNet Link table.
2. Initial constraints returned by MetaNet and stored in the Link table.
3. RequestChannel() call received by CRP from an external client.
4. Channel table populated with pre-allocation channel request information.
5. GetImportance() call to IPM made.
6. Importance value returned. This value is mapped to a MetaNet priority. Additional shaping of the request's Utility function can take place at this point, to form a "conditioned" channel request. Presently, we do not shape the utility function.
7. Channel table updated to reflect the priority assignment.
8. Policy Manager passes the conditioned channel request to the RA.
9. GetAvaliableEndToEndCapacity() called by AIC to initialize the RA's capacity table. This ensures up to date link capacity data.
10. Link capacity information returned by MetaNet and stored in capacity table.
11. Allocate() called on the channel allocation algorithm to determine admission decision. If accepted, preemptable channels and modifiable channels are also determined.
12. ModifyChannels() called if the channel was accepted, and existing channels must be modified to accommodate the new channel.
13. SetupMetaNetChannel() call to MetaNet. This call also supplies the preemptable channel list.
14. Return from SetupMetaNetChannel().
15. Update channel table with the "provisioned" QoS.
16. Return to external client with channel admissibility information.

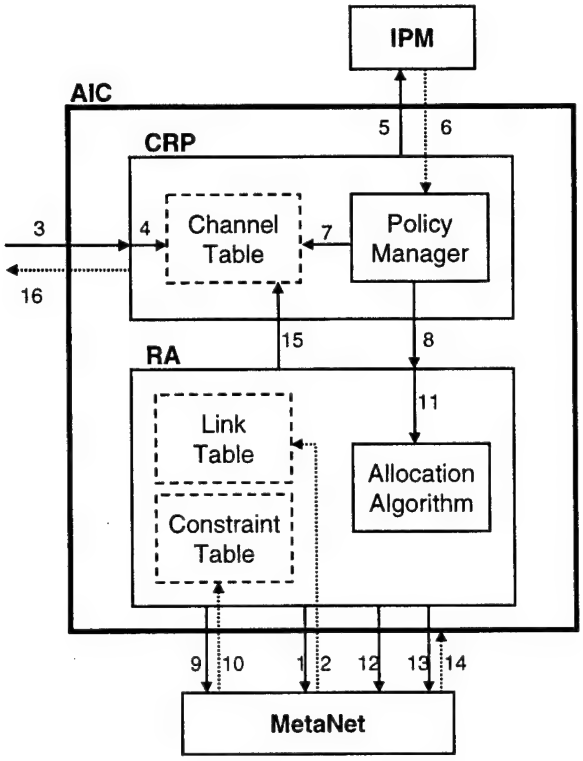


Figure 3 - Allocation function call sequence

1.5 Results

AIC has been tested under various scenarios. For purposes of illustration, we describe a set of simple test cases to demonstrate AIC's channel allocation characteristics. The MetaNet topology used for all test cases was:

- Three hostgroups: "hostgroup0", "hostgroup1", and "hostgroup2", with one host per hostgroup.
- hostgroup0 - hostgroup1 & hostgroup0 - hostgroup2 are *Dependent* links.
- Maximum channel capacity for each link is 600 bandwidth units.

We exercise the following AIC channel allocation characteristics:

- Channel rejection by optimizer
- Channel preemption by optimizer
- Priority-based Preemption
- Use of Dependent links
- Use of multi-choice Utility function
- Modifying QoS of allocated channel

Each characteristic is demonstrated below. Each request is stated, followed by the result in the form of a channel table listing. The requests are of the form:

Request<#>: <*Hostgroup*>→< *Hostgroup*>, p = <*priority*>, {BW =<*bandwidth*>, u =<*utility*>}$^{+}$

The results are of the form:

Result: <*accept or reject*>
Preemptable: <*list of AIC channel numbers*>
Channel Table: <*set of channels*>

The requests all overlap in time in order that they compete with each other for resources, so the specific time and duration parameters are not shown.

1.5.1 Rejection by Optimizer

Here the optimizer will determine that the aggregate utility of the existing allocated channels is better than that of the new request.

Request1: 0→1, p = 6, BW = 300, u = 0.3
Result: Accepted
Preemptable: None
Channel Table: AIC1, P:6, U:0.30, Bw:300.00

Request2: 0→1, p = 6, BW = 300, u = 0.2
Result: Accepted
Preemptable: None
Channel Table: AIC1, P:6, U:0.30, Bw:300.00
AIC2, P:6, U:0.20, Bw:300.00

Request3: 0→1, p = 6, BW = 500, u = 0.4
Result: Rejected by optimizer
Preemptable: None
Channel Table: AIC1, P:6, U:0.30, Bw:300.00
AIC2, P:6, U:0.20, Bw:300.00

1.5.2 Preemption by Optimizer

Here the optimizer will use the utility of the requested channel to determine that the single utility of the new request is better than that of the existing allocated channels.

Request1: 0→1, p = 6, BW = 300, u = 0.3
Result: Allocated
Preemptable: None
Channel Table: AIC1, P:6, U:0.30, Bw:300.00

Request2: 0→1, p = 6, BW = 300, u = 0.2
Result: Allocated
Preemptable: None
Channel Table: AIC1, P:6, U:0.30, Bw:300.00
AIC2, P:6, U:0.20, Bw:300.00

Request3: 0→1, p = 6, BW = 500, u = 0.6
Result: Accepted
Preemptable: 1,2
Channel Table: AIC3, P:6, U:0.60, Bw:500.00

1.5.3 Priority-based Preemption

This example allocates 'low' priority channels to fill a link, and then issues a request with a 'high' priority, to force the RA to recommend some channels to preempt.. The link capacity is 600 units. Note: The utility of each request is ignored by priority-based preemption. The RA will preempt only the channels necessary to free the required number of resources.

Request1: 0→1, p = 3, BW = 200, u = 0.3
Result: Accept
Preemptable: None
Channel Table: AIC1, P:3, U:0.30, Bw:200.00

Request2: 0→1, p = 4, BW = 200, u = 0.3
Result: Accept
Preemptable: None
Channel Table: AIC1, P:3, U:0.30, Bw:200.00
AIC2, P:4, U:0.30, Bw:200.00

Request3: 0→1, p = 5, BW = 200, u = 0.3
Result: Accept
Preemptable: None
Channel Table: AIC1, P:3, U:0.30, Bw:200.00
AIC2, P:4, U:0.30, Bw:200.00
AIC3, P:5, U:0.30, Bw:200.00

Request4: 0→1, p = 2, BW = 200, u = 0.3
Result: Accept
Preemptable: 3
Channel Table: AIC1, P:3, U:0.30, Bw:200.00
AIC2, P:4, U:0.30, Bw:200.00
AIC4, P:2, U:0.30, Bw:200.00

1.5.4 Use of Dependent Links

The RA considers all the temporally overlapping channels on dependent links to determine if a request should be allocated. This is conservative, since not all channels on dependent link are 'additive' in their capacity consumption. However, it ensures that MetaNet will always accept a channel setup request from AIC.

Request1: 0→1, p = 3, BW = 300, u = 0.3

Result: Accept
Preemptable: None
Channel Table: AIC1, P:3, U:0.30, Bw:300.00

Request2: 0→2, p = 3, BW = 300, u = 0.3
Result: Accept
Preemptable: None
Channel Table: AIC1, P:3, U:0.30, Bw:300.00
AIC2, P:3, U:0.30, Bw:300.00

Request3: 0→1, p = 2, BW = 400, u = 0.3
Result: Accept
Preemptable: 1,2
Channel Table: AIC3, P:2, U:0.30, Bw:400.00

1.5.5 Use of Multiple-utilities

This example shows a request with multiple choices for its QoS being admitted at a Qos level that maximizes the overall utility at the requested priority level. Initially two requests are admitted and accepted at priority 6. The third request contains a 3-point Qos region. The optimizer selects the highest utility point in the region at the expense of the two existing channels due to its higher utility.

Request1: 0→1, p = 6, BW = 300, u = 0.3
Result: Accepted
Preemptable: None
Channel Table: AIC1, P:6, U:0.30, Bw:300.00

Request2: 0→1, p = 6, BW = 300, u = 0.2
Result: Accepted
Preemptable: None
Channel Table: AIC1, P:6, U:0.30, Bw:300.00
AIC2, P:6, U:0.20, Bw:300.00

Request3: 0→1, p = 6, {BW = 300, u = 0.4}, {BW = 400, u = 0.5}, {BW = 500, u = 0.6}
Result: Accepted
Preemptable: 1,2
Channel Table: AIC3, P:6, U:0.60, Bw:500.00

1.5.6 Modifying Qos of allocated channel

Continuing from the last test, a request with multiple choices for its QoS is admitted at a QoS level that maximizes the overall utility at the requested priority level. The optimizer also selects a new operating point for an existing channel.

Request4: 0→1, p = 6, {BW = 300, u = 0.4}, {BW = 400, u = 0.5}, {BW = 500, u = 0.6}
Result: Accepted

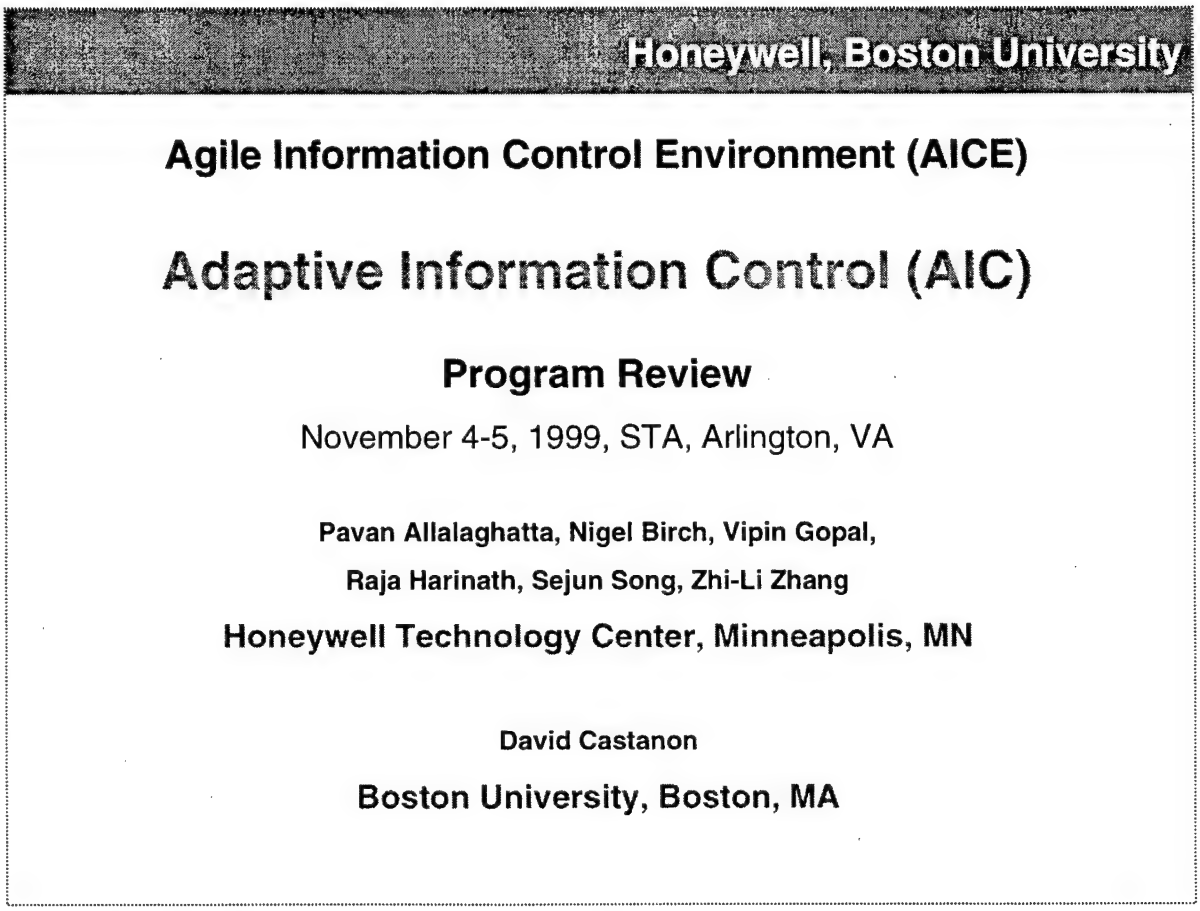
Preemptable: 1, 2
Channel Table: AIC3, P:6, U:0.40, Bw:300.00
AIC4, P:6, U:0.40, Bw:300.00

1.6 Summary

This report has described the Adaptive Information Controller (AIC) component of the Agile Information Control Environment (AICE). AIC implements a communication channel resource allocation algorithm that allocates channels according to their priority, and optimizes the delivered utility of the admitted channels. Example allocations have been described to illustrate the characteristics of the channel allocation algorithm. The current approach has considered only one QoS dimension, i.e., bandwidth. As future work, we plan to develop allocation algorithms for multi-dimensional QoS problems, and predictive models for admission control and resource allocation [Godb99].

Section 2 Adaptive Information Control (AIC) Layer

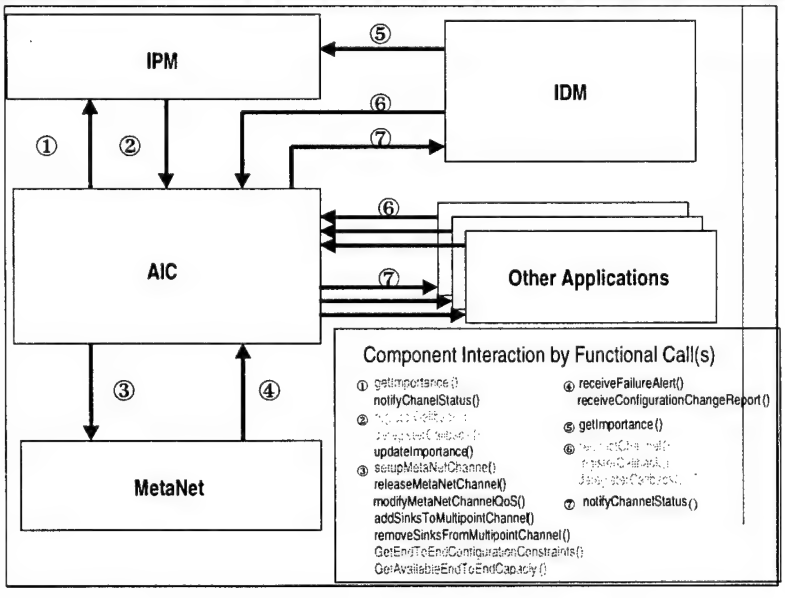
The following are slides from a presentation made at the DARPA PI meeting in Washington, DC, in November 1999.



Agenda

- Year I Plan
- Current Status
- AIC Overview
- AIC Implementation
- Plans for the remainder of contract period

Year I Plan



AICE PAC99 Architecture and Interfaces

Year I Plan

Interface	Implemented by	Called by
getImportance	IPM	AIC
notifyChannelStatus	IPM/IDM/External Sys.	AIC
requestChannel	AIC	IDM/External Systems
registerCallBack	AIC	IPM/IDM/External Sys.
deregisterCallBack	AIC	IPM/IDM/External Sys.
SetupMetaNetChannel	MetaNet	AIC
GetEndToEndConfigurationConstraints	MetaNet	AIC
GetAvaliableEndToEndCapacity	MetaNet	AIC

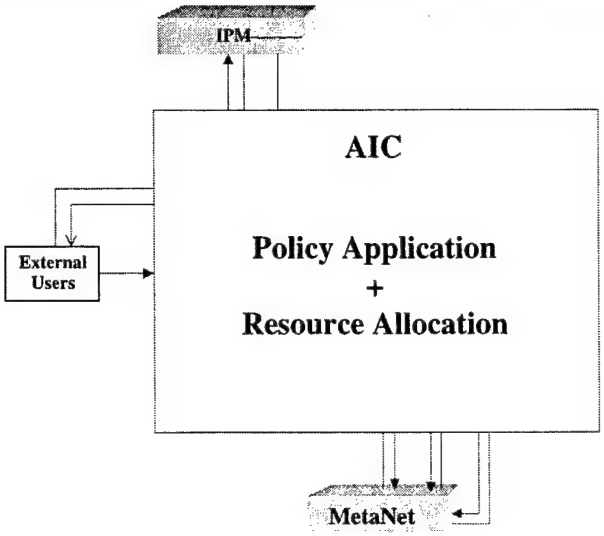
Phase 1 External Interfaces

- AIC layer will support two major functional components
  - CRP - Channel Request Processing
  - RA - Resource Allocator

Current Status - Milestones

- Major AIC components for Year I implemented (Sep 30, 1999)
- Unit test for AIC components completed at TRW (Oct 5, 1999)
- Boston University (BU) RA added (Oct 14, 1999)
- Unit test for AIC with BU RA completed (Oct 18, 1999)
- AIC Integration completed at TRW (Oct 19, 1999)
- Integration testing completed at TRW (Oct 29, 1999)

Overview - AIC and External Components



External Simulator - Overview

- Implements a simple model of the External Clients
- Design goal: Sufficient to perform AIC unit tests
- Implements a part of the ExternalToAic interface
  - Aice::ChannelName requestChannel(
    in Aice::SetOfOwnerNames w,
    in Aice::ApplicationLocation s,
    in Aice::SetOfApplicationLocations d,
    in Aice::ChannelExchangeCharacterization c,
    in AicDataType::UtilityFunction u);

External Simulator - Overview

- Inputs to AIC: from the console, file, and random.
- Input data format (from the data file):
  - timezero = 10/7/1999 19:00:00 EST
  - channel = 1
  - requestTime = 0
  - source = ehf_satcom://host0:10
  - dest = ehf_satcom://host1:7
  - owner = 6
  - exchar = Intel:IMINT:EO
  - utility = **0.4**
  - requestQoS = **BANDWIDTH**
  - bw = **300**
  - startTime = 30
  - endTime = 330
  - duration = 120
  - delay = 0.3
  - loss = 6
  - endutility
  - endchannel

External Simulator - Overview

- Example of a request:

```
processing file
 The Owner is : 6
 The Source address is : ehf_satcom://host0:10
 The Destination address is : ehf_satcom://host1:7
 The start time is =  1999:10:7:19:0:30
 The end time is =  1999:10:7:19:5:30
 The requested bandwidth is : 300
 The Utility value is : 0.4
```

- Example of return values:
  - If successful:
    - `returned value was 1`
  - If failed:
    - `returned value was NULL`

IPM Simulator – Overview

- Implements a simple model of IPM
- Design goal: Sufficient to perform AIC unit tests
- Implements a part of the AicToIpm interface
  - Aice::Importance getImportance(
    in Aice::SetOfOwnerNames w,
    in Aice::ApplicationLocation s,
    in Aice::SetOfApplicationLocations d,
    in Aice::ChannelExchangeCharacterization c,
    in Aice::ChannelName channelId);
- Returns with a random number between 0.0-1.0

MetaNet Simulator – Overview

- Implements a simple model of the MetaNet
- Design goal: Sufficient to perform AIC unit tests
- For Year I: Implements a part of the AicToMetanet interface
  - setupMetanetChannel()
  - releaseMetanetChannel()
  - GetEndToEndConfigurationConstraints()
  - GetAvailableEndToEndCapacity()

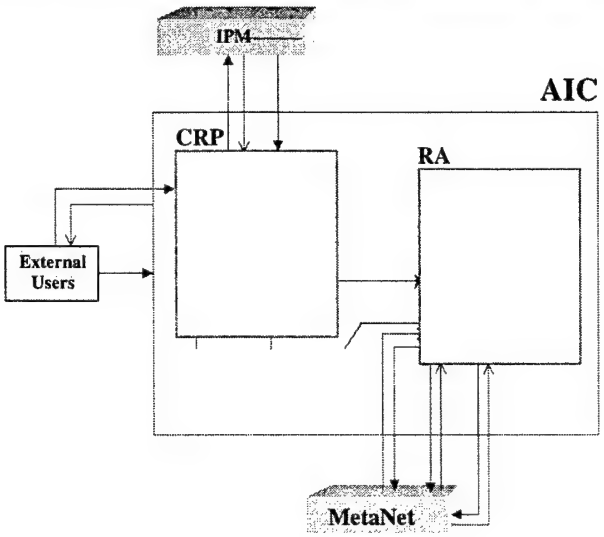
Metanet Simulator - Assumptions

- All links are considered point-to-point
  - no point-to-multipoint
- Supports multiple hosts per hostgroup
  - current scenario: just one host per hostgroup
- Supports hosts belonging to multiple hostgroups
  - current scenario: a host belongs to a single hostgroup
- No explicit topology
  - simplifies implementation
  - provides enough information for AIC to operate
  - topology implied in dependent link information
    - use common link bandwidth constraint to model dependent link behavior

Metanet Simulator - Implementation

- Maintains detailed (to the second) information about bandwidth allocations
  - separate information for links and dependent links
  - GetAvailableEndToEndCapacity combines both information
- setupMetanetChannel preempts only the channels listed in expendable channel list
  - for repeatable experiments
    - AIC does not depend on this behavior
  - can preempt channels even if request rejected
    - simplifies implementation

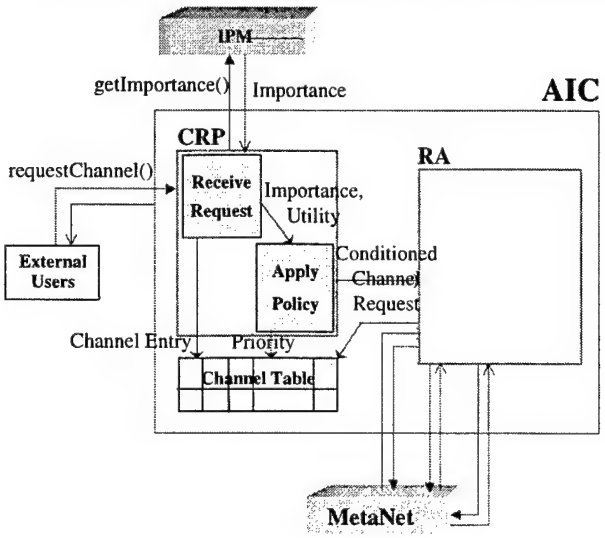
Overview – AIC Components



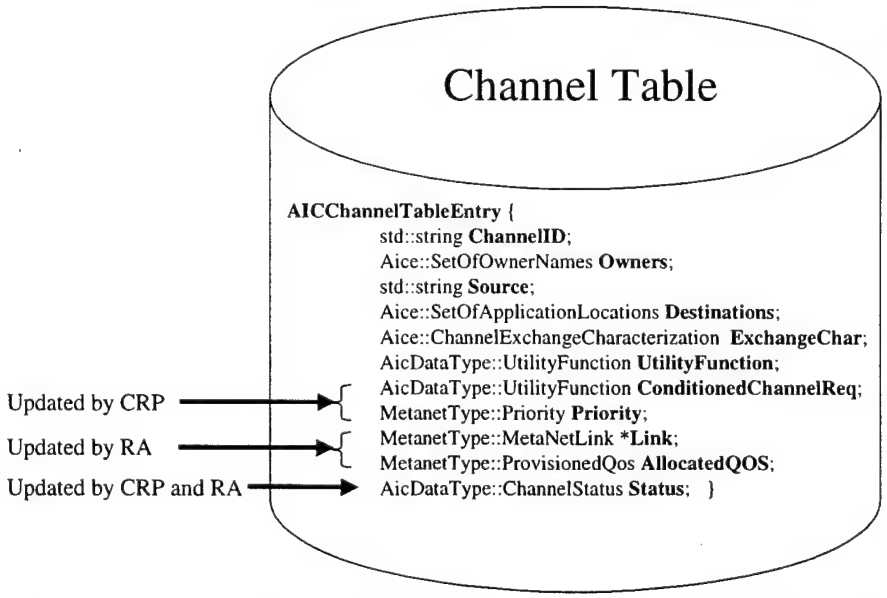
Channel Request Processor (CRP)

- Accepts *requestChannel* from IDM/External
- Generates the *Conditioned_Channel_Request* to Resource Allocator
- Consists of two components:
  - Receive Request (RR):
    - Receives request from the external systems
    - Populates the channel table
    - Makes *getImportance* call to IPM
  - Apply Policy (AP):
    - Receives the returned importance from IPM
    - Maps the importance to priority and shapes the utility function
    - Updates the channel table
    - Generates Conditioned_Channel_Request to Resource Allocator

Channel Request Processor (CRP) - Control Flow



Channel Request Processor (CRP)



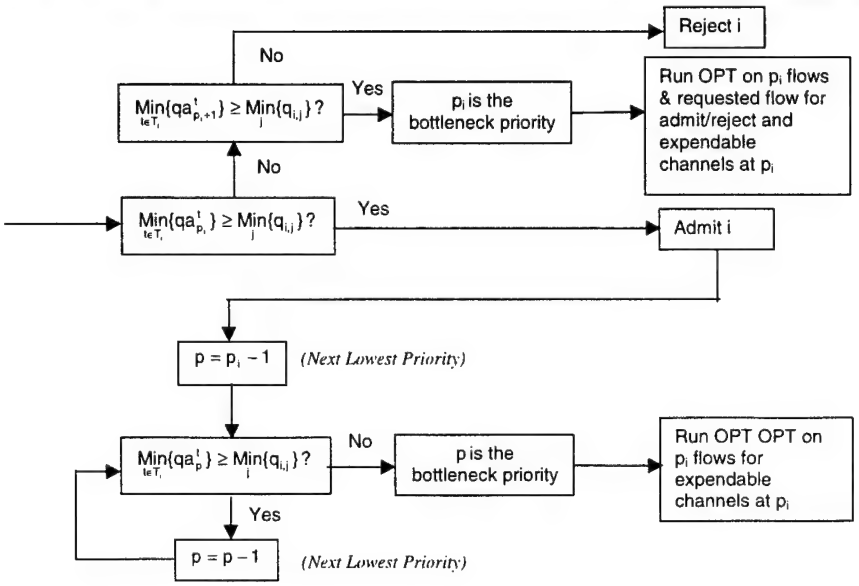
Resource Allocation Problem

- Problem statement:
  - A request for setting up a channel i for link L from {t0, tf} is received. The request comes with a utility function that maps QoS allocations to the corresponding utility. The Admission Control and Resource Allocation (ACRA) problem is to compute:
    - Whether channel i will be admitted.
    - If yes, the amount of resources that will be allocated to it and the set of allocated channels that need to be preempted/degraded for admitting i.
- Assumptions:
  - A higher priority flow cannot be rejected if the requested resources can be freed up by preempting lower priority flows.
  - The following information is known:
    - Maximum channel capacity.
    - Available link capacity per priority for the time intervals for which the request is being made.
    - Link dependency information.

Proposed Solution

- Definitions:
  - Request i is of the form $r_i(L_i, p_i, \{u_{ij}, q_{ij}\}, t_i,0, t_i,f)$.
    - L_i is the corresponding link
    - p_i is the priority of the request.
    - $\{u_{ij}, q_{ij}\}$s are the set of discrete points which maps the QoS parameters, q, (here, q is bandwidth) to their utility u.
    - $t_i,0$, t_i,f are the start and end times of the request.
  - q_{apt} is the Available bandwidth in time interval t at priority p.
  - Bottleneck priority level (p^b_i)
    - The highest priority level where there are not enough resources available to satisfy even the minimum requested resources for i.
- Basic Approach
  - Determine the bottleneck priority.
  - Optimize flows at bottleneck priority. This may determine if the request is accepted or rejected.
  - Flows of a lower lower priority than the bottleneck AND those discarded by optimizer are preemptable.

Derivation of Bottleneck Priority – HTC



Optimization at Bottleneck Priority

- **Flexible knapsack problem (FKP).**
- Objective: maximizing the overall utility
- Constraint: bounded bandwidth allocation.

$$Max \sum_i \sum_j y_{i,j} u_{i,j}$$

$$s.t. \sum_i \sum_j y_{i,j} q_{i,j} \le qa_p$$

$$\sum_j y_{i,j} \le 1 \quad \forall i$$

$$y_{i,j} \in \{0,1\}$$

- **0-1 knapsack problem (KP).**
- A variation omits j, i.e., no flexibility in the value of the QoS parameter.
  - We have implemented this for AICE

- **Adding a Temporal Component**
- Changing Utility with time completed.
- alpha is the fraction of the time completed.
  - Weights flows near completion with greater utility.

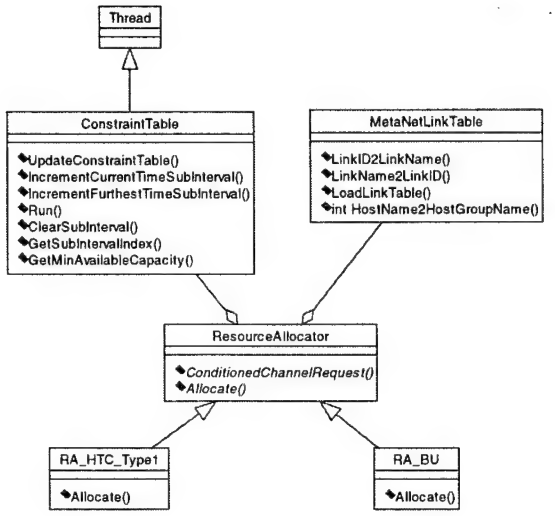
$$Max \sum_{i \in M_p} \sum_j M\alpha y_{i,j} u_{i,j} + \sum_{i \in L_p} \sum_j y_{i,j} u_{i,j}$$

$$s.t. \sum_i \sum_j y_{i,j} q_{i,j} \le qa_p$$

$$\sum_j y_{i,j} \le 1 \quad \forall i$$

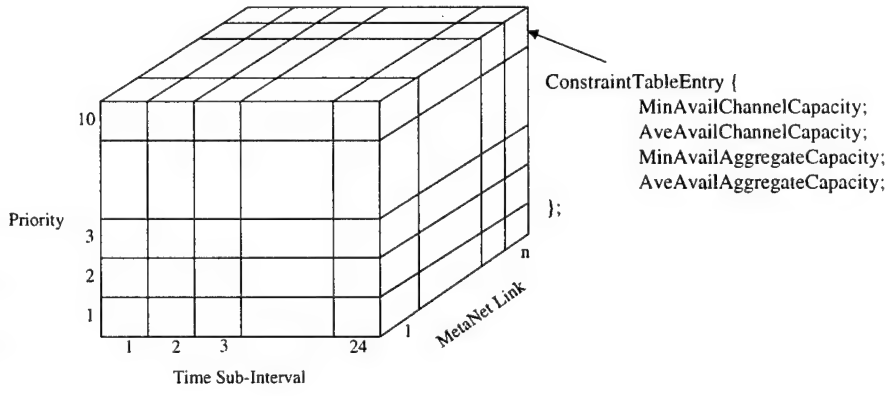
$$y_{i,j} \in \{0,1\}$$

Resource Allocator Implementation



Resource Allocator Constraint Table

- Used by the Resource Allocator algorithm to determine admissibility and channel preemption.
- Maintained via `GetEndToEndAvailableCapacity()` calls to MetaNet.



Comparison of Resource Allocators

	HTC RA	BU RA
QOS Parameter	Bandwidth. Minimum QoS bandwidth element.	Bandwidth. Uses all QoS vector elements.
Use of Priority	Channels of lower priority are candidates for preemption.	All channels of lower priority are preempted.
Use of Utility	Utility used to choose among channels of the bottleneck priority.	Utility used to choose among channels at the request priority.
Optimization Method	Knapsack formulation. • Objective Function maximizes aggregate utility. • Constraint function bounds the bandwidth allocation.	Branch and bound scheduler.
Optimizer Input channel set	If request can be satisfied at requested Priority, P: • Set of allocated channels at the Bottleneck priority. If request cannot be satisfied at requested Priority, but can at the next higher priority: • Set of allocated channels at requested priority, including the requested channel.	Channels at the requested priority on the requested Link.
Dependent Links used?	Yes. Conservative allocations, i.e., MetaNet will never reject an allocation. RA may reject when there is bandwidth available.	No. Aggressive allocations. MetaNet may reject some allocations due to insufficient bandwidth.
Optimizer output	Set of channels to be preempted at bottleneck priority + lower priority channels.	Set of channels to be preempted at requested priority + lower priority channels.

Resource Allocation Examples

- The tests were run using the HTC MetaNet simulator which simulates the following network topology:
  - 3 HostGroups: hostgroup0, hostgroup1, and hostgroup2. There is one Host per hostgroup.
  - hostgroup0→ hostgroup1 and hostgroup0→ hostgroup2 are *Dependent* links.
  - Maximum initial Channel Capacity for each link is 600 bandwidth units.

Rejection by Optimizer

1.1 Request 1

Request: 0→1, t = [BT+0:30 to BT+5:30], p = 6, min BW = 300, u = 0.3

	HTC RA	BU RA
Result	Accepted due to no prior allocations	Accepted due to no prior allocations
Expendable Channels	None	None
Channel Table	AIC1 => MNChannel0, P:6, U:0.30, Bw:300.00	AIC1 => MNChannel0, P:6, U:0.30, Bw:300.00

1.2 Request 2

Request: 0→1, t = [BT+0:30 to BT+5:30], p = 6, min BW = 300, u = 0.2

	HTC RA	BU RA
Result	Accepted	Accepted
Expendable Channels	None	None
Channel Table	AIC1 => MNChannel0, P:6, U:0.30, Bw:300.00 AIC2 => MNChannel1, P:6, U:0.20, Bw:300.00	AIC1 => MNChannel0, P:6, U:0.30, Bw:300.00 AIC2 => MNChannel1, P:6, U:0.20, Bw:300.00

1.3 Request 3

Request: 0→1, t = [BT+0:30 to BT+5:30], p = 6, min BW = 500, u = 0.4

	HTC RA	BU RA
Result	Rejected by optimizer	Rejected by optimizer
Expendable Channels	None	None
Channel Table	AIC1 => MNChannel0, P:6, U:0.30, Bw:300.00 AIC2 => MNChannel1, P:6, U:0.20, Bw:300.00	AIC1 => MNChannel0, P:6, U:0.30, Bw:300.00 AIC2 => MNChannel1, P:6, U:0.20, Bw:300.00

Preemption by Optimizer

1.1 Request 1

Request: 0→1, t = [BT+0:30 to BT+5:30], p = 6, min BW = 300, u = 0.3

	HTC RA	BU RA
Result	Allocated	Allocated
Expendable Channels	None	None
Channel Table	AIC1 => MNChannel0, P:6, U:0.30, Bw:300.00	AIC1 => MNChannel0, P:6, U:0.30, Bw:300.00

1.2 Request 2

Request: 0→1, t = [BT+0:30 to BT+5:30], p = 6, min BW = 300, u = 0.2

	HTC RA	BU RA
Result	Allocated	Allocated
Expendable Channels	None	None
Channel Table	AIC1 => MNChannel0, P:6, U:0.30, Bw:300.00 AIC2 => MNChannel1, P:6, U:0.20, Bw:300.00	AIC1 => MNChannel0, P:6, U:0.30, Bw:300.00 AIC2 => MNChannel1, P:6, U:0.20, Bw:300.00

1.3 Request 3

Request: 0→1, t = [BT+0:30 to BT+5:30], p = 6, min BW = 500, u = 0.6

	HTC RA	BU RA
Result	Accepted	Accepted
Expendable Channels	MNChannel0 MNChannel1	MNChannel0 MNChannel1
Channel Table	AIC3 => MNChannel2, P:6, U:0.60, Bw:500.00	AIC3 => MNChannel2, P:6, U:0.60, Bw:500.00

Priority-based Preemption

1.1 Request 1

Request: 0→1, p = 3, min BW = 200, u = 0.3

	HTC RA	BU RA
Result	Accept	Accept
Expendable Channels	None	None
Channel Table	AIC1 => MNChannel:0, P:3, U:0.30, Bw:200.00	AIC1 => MNChannel:0, P:3, U:0.30, Bw:200.00

1.2 Request 2

Request: 0→1, p = 4, min BW = 200, u = 0.3

	HTC RA	BU RA
Result	Accept	Accept
Expendable Channels	None	None
Channel Table	AIC1 => MNChannel:0, P:3, U:0.30, Bw:200.00 AIC2 => MNChannel1, P:4, U:0.30, Bw:200.00	AIC1 => MNChannel:0, P:3, U:0.30, Bw:200.00 AIC2 => MNChannel1, P:4, U:0.30, Bw:200.00

1.3 Request 3

Request: 0→1, p = 5, min BW = 200, u = 0.3

	HTC RA	BU RA
Result	Accept	Accept
Expendable Channels	None	None
Channel Table	AIC1 => MNChannel:0, P:3, U:0.30, Bw:200.00 AIC2 => MNChannel1, P:4, U:0.30, Bw:200.00 AIC3 => MNChannel2, P:5, U:0.30, Bw:200.00	AIC1 => MNChannel:0, P:3, U:0.30, Bw:200.00 AIC2 => MNChannel1, P:4, U:0.30, Bw:200.00 AIC3 => MNChannel2, P:5, U:0.30, Bw:200.00

1.4 Request 4

Request: 0→1, p = 2, min BW = 200, u = 0.3

	HTC RA	BU RA
Result	Accept	Accept
Expendable Channels	MNChannel2	MNChannel0 MNChannel1 MNChannel2
Channel Table	AIC1 => MNChannel0, P:3, U:0.30, Bw:200.00 AIC2 => MNChannel1, P:4, U:0.30, Bw:200.00 AIC4 => MNChannel3, P:2, U:0.30, Bw:200.00	AIC2 => MNChannel1, P:4, U:0.30, Bw:200.00 AIC3 => MNChannel2, P:5, U:0.30, Bw:200.00 AIC4 => MNChannel3, P:2, U:0.30, Bw:200.00

Use of Dependent Links

1.1 Request 1

Request: 0→1, p = 3, min BW = 300, u = 0.3

	HTC RA	BU RA
Result	Accept	Accept
Expendable Channels	None	None
Channel Table	AIC1 => MNChannel0, P:3, U:0.30, Bw:300.00	AIC1 => MNChannel0, P:3, U:0.30, Bw:300.00

1.2 Request 2

Request: 0→2, p = 3, min BW = 300, u = 0.3

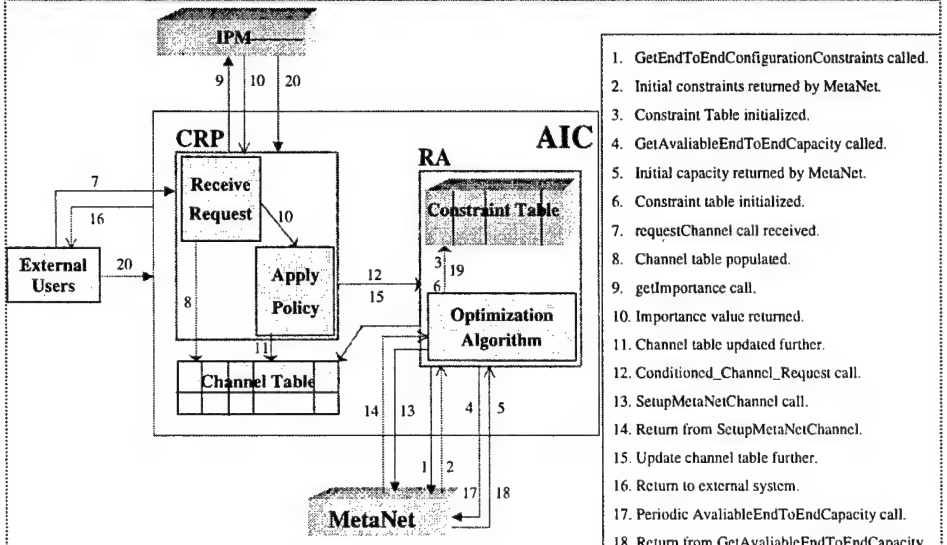
	HTC RA	BU RA
Result	Accept	Accept
Expendable Channels	None	None
Channel Table	AIC1 => MNChannel0, P:3, U:0.30, Bw:300.00 AIC2 => MNChannel1, P:3, U:0.30, Bw:300.00	AIC1 => MNChannel0, P:3, U:0.30, Bw:300.00 AIC2 => MNChannel1, P:3, U:0.30, Bw:300.00

1.3 Request 3

Request: 0→1, p = 2, min BW = 400, u = 0.3

	HTC RA	BU RA
RA Result	Accept	Accept
Expendable Channels	MNChannel0 MNChannel1	MNChannel0
MN Result	Accept	Rejected
Channel Table	AIC3 => MNChannel2, P:2, U:0.30, Bw:400.00	AIC1 => MNChannel0, P:3, U:0.30, Bw:300.00 AIC2 => MNChannel1, P:3, U:0.30, Bw:300.00

Summary - AIC Control Flow



Further Work (11/99 - 01/99)

- Support experimental and evaluation work for PAC99.
- Continue unit testing and evaluation at HTC.
- Theoretical analysis of more complex admission control and resource allocation problems.
- Support a temporal component for utility function.
  - Allocated channels have greater value as they near completion.
- Support point-to-multipoint channel requests.
- Support 'batch' requests.
  - Multiple channels are allocated with a single request.

Additional Knapsack Problem Formulations

- **Multidimensional knapsack problem (FKP)**
- K is the set of all Qos parameters considered
- The utility can be a function of multiple QoS parameters such as bandwidth, delay and jitter.
- In this case, then we have a **multidimensional** constraint, each dimension corresponding to a QoS parameter.

$$Max \sum_i y_i u_i$$

$$s.t. \quad \sum_i y_i q_i^k \le qa_p^k \quad k \in K$$

$$y_i \in \{0,1\}$$

- **Flexible multidimensional knapsack problem (FMDKP)**
- Here each Qos parameter can have multiple values each with its own utility.

$$Max \sum_i \sum_j y_{i,j} u_{i,j}$$

$$s.t. \quad \sum_i \sum_j y_{i,j} q_{i,j}^k \le qa_p^k \quad k \in K$$

$$\sum_j y_{i,j} \le 1 \quad \forall i$$

$$y_{i,j} \in \{0,1\}$$

Knapsack Problem Classifications

- **0-1 knapsack problem (KP).**
  - One QoS parameter with one associated utility.
    - Note: We have implemented this for AIC
- **Flexible knapsack problem (FKP).**
  - The QoS parameter has 'flexibility' in its assignment. A separate utility is applied to each QoS parameter value.
  - The channel has the flexibility to be allocated at any one of the discrete points
- **Multidimensional knapsack problem (FKP)**
  - The utility can be a function of multiple QoS parameters such as bandwidth, delay and jitter.
  - In this case, then we have a multidimensional constraint, each dimension corresponding to a different QoS parameter.
- **Flexible multidimensional knapsack problem (FMDKP)**
  - Here each QoS parameter can have multiple values each with its own utility.

Section 3 Stochastic Flow Placement and Admission Control

3.1 Assumptions about the Flows

We assume that flows can be classified into "types'. There are total of K types. For each type i, flows of type i can have a discrete set I_i of possible bandwidth allocations. For a bandwidth allocation of $c_{i,j}$, where $j \in I_i$, the associated utility for a flow of type i with this bandwidth allocation is $u_{i,j}$.

Consider a "typical" time interval τ. Let λ_i be the "average" number of flows of type i arriving during the time interval T. In Section 6, we will discuss how these terms can be specified more precisely and obtained in practice. For simplicity of exposition, we will only consider the case where no priority is associated with flows. Later on, we will discuss how priority can be incorporated (see Section 3.5).

In the following we will consider the problems of 1) maximizing the expected number of flows that can be admitted in the system over a "typical" time interval τ; and 2) maximizing the expected utility of flows that can be admitted in the system over a typical time interval τ. The optimization problem formulation is presented in Section 3.2. This formulation is extended in Section 3.3 where we address the issue of admission control at a flow arrival instant. In Section 3.4 we consider the problem of admission control for flows with finite duration.

3.2 Optimal Flow Placement

We first consider the problem of maximizing the expected number of flows that can be admitted into a system over a "typical" time interval τ, given that the average number of flows of type I during the time interval τ is λ_i. The total capacity of the system is assumed to be C. This optimization problem is formulated as the following linear programming problem.

3.2.1 Problem Formulation A.1: Maximizing Expected Number of Flows

$$\text{maximize} \quad \sum_{i=1}^{K} x_i$$

$$\text{subject to} \quad x_i = \sum_{j \in I_i} x_{i,j} \le \lambda_i, \; i = 1,2,...,K$$

$$\sum_{i=1}^{K} \sum_{j \in I_i} x_{i,j} c_{i,j} \le C$$

$$\text{and } x_{i,j} \ge 0 \;\; i = 1,2,...,K, \text{ and } j \in I_i$$

Let $\{ x_i^* \}$, $i = 1,2,...,K$ be the solutions to the above optimization problem, where in particular $x_i^* = \sum_{j \in I_i} x_{i,j}^*$, and $x_{i,j}^* \geq 0$ is the maximal (expected) number of flows of type i that can be admitted into the system with a bandwidth allocation of $c_{i,j}$.

Assume that for each flow type i, $c_{i,1} \leq c_{i,j}$ for any $j \in I_i$. Then it is not too hard to see that only $x_{i,1}^*$ can be nonzero. In other words, for $j \in I_i$ and $j \neq 1$, $x_{i,j}^* = 0$. Intuitively, we can always admit more flows into the system by allocating only the minimum bandwidth requested by the flows. Hence in this optimization formulation, we do not consider the utility of the bandwidth allocation accorded to a flow. In the following consider the problem of maximizing the expected utility of flows that can be admitted into a system over a "typical" time interval τ, given that the average number of flows of type I during the time interval τ is λ_i. This optimization problem is again formulated as a linear programming problem, as shown below.

3.2.2 Problem Formulation A.2: Maximizing Expected Utility

$$\text{maximize} \quad \sum_{i=1}^{K} \sum_{j \in I_i} x_{i,j} u_{i,j}$$

$$\text{subject to} \quad x_i = \sum_{j \in I_i} x_{i,j} \leq \lambda_i, \; i = 1,2,...,K$$

$$\sum_{i=1}^{K} \sum_{j \in I_i} x_{i,j} c_{i,j} \leq C$$

$$\text{and} \quad x_{i,j} \geq 0, \; i = 1,2,...,K, \text{ and } j \in I_i$$

As before, let $\{ x_i^* \}$, $i = 1,2,...,K$ be the solutions to the above optimization problem, where in particular $x_i^* = \sum_{j \in I_i} x_{i,j}^*$, and $x_{i,j}^* \geq 0$ is the expected number of flows of type i with a bandwidth allocation of $c_{i,j}$. We can design a simple probabilistic flow admission control algorithm using the optimal solutions (this algorithm also applies to the Optimization Problem Formulation A.1):

> *A flow of type i is admitted into the system with probability $p_i = \dfrac{\sum_{j \in I_i} x_i^*}{\lambda_i}$, and assigned a bandwidth allocation of $c_{i,j}$ with probability. Furthermore, we may impose an additional condition that a flow of type i may be rejected if the total number of flows of type i that have been admitted into the system reaches λ_i.*

3.3 Flow Admission Control with Memory

In the previous section, we consider the optimal flow placement to maximize either the expected number of flows admitted into the system or the expected utility of flows admitted into the system. The solutions to the aforementioned optimization problems lead, to a simple probabilistic flow admission control algorithm. This simple algorithm, however, does not take any "history" into account: upon arrival of a flow of type i, it is considered admission with probability $p_i = \frac{\sum_{j \in I_i} x_i^*}{\lambda_i}$, and considered a bandwidth allocation of $c_{i,j}$ with probability $p_{i.j} = \frac{x_{i,j}^*}{x_i^*}$. The number of flows that are currently admitted in the system is not taken into account. In the following, we improve this simple admission control by "remembering the history". There are two ways we can utilize the "history" information.

3.3.1 Method 1: Use of Residual Bandwidth

We use the *residual* bandwidth C_{res} of the system as the limiting constraint to determine the expected number of "new" flows that can be admitted into the system (in the "next typical time interval"). Here C_{res} equals to the difference between C and the total bandwidth allocated to the existing flows. In this method, we implicitly assume that flow arrivals are independent and "memoryless" (i.e., Poisson): the "average" (or "expected") number of flows arriving in the "next typical time interval" is still λ_i. In addition, there are no flow departures in the time interval. The optimization problem formulation is presented below.

3.3.1.1 Problem Formulation B.1.1: Maximizing Expected Number of Flows

$$\text{maximize} \quad \sum_{i=1}^{K} z_i$$

$$\text{subject to} \quad z_i = \sum_{j \in I_i} z_{i,j} \le \lambda_i, \; i = 1,2,...,K$$

$$\sum_{i=1}^{K} \sum_{j \in I_i} z_{i,j} c_{i,j} \le C_{res}$$

$$\text{and} \quad z_{i,j} \ge 0, \; i = 1,2,...,K, \text{ and } j \in I_i$$

3.3.1.2 Problem Formulation B.1.2: Maximizing Expected Utility

$$\text{maximize} \quad \sum_{i=1}^{K} \sum_{j \in I_i} z_{i,j} u_{i,j}$$

$$\text{subject to} \quad z_i = \sum_{j \in I_i} z_{i,j} \le \lambda_i, \; i = 1,2,...,K$$

$$\sum_{i=1}^{K}\sum_{j\in I_i} x_{i,j}c_{i,j} \le C_{res}$$

and $z_{i,j} \ge 0$, $i = 1,2,...,K$, and $j \in I_i$

3.3.1.3 Solutions

Let $\{ z_i^* \}$, $i = 1,2,...,K$ be the solutions to the either of the two optimization problems, where in particular $z_i^* = \sum_{j\in I_i} z_{i,j}^*$, and $z_{i,j}^* \ge 0$ is the expected number of flows of type i with a bandwidth allocation of $c_{i,j}$. We have the following probabilistic flow admission control algorithm.

> *A flow of type i is admitted into the system with probability* $p_i = \dfrac{\sum_{j\in I_i} z_i^*}{\lambda_i}$,
>
> *and is allocated $c_{i,j}$ amount of bandwidth with probability* $p_{i,j} = \dfrac{z_{i,j}^*}{z_i^*}$.
>
> *Furthermore, we may impose an additional condition that a flow of type i may be rejected if the total number of flows of type i that have been admitted into the system reaches λ_i.*

Comparing the above flow admission control algorithm with the one at the end of Section 2, we see that by replacing C with C_{res}, the optimal solutions are affected by the existing flows in the system. In general, when there are more flows existing in the system, fewer flows will likely be considered for admission and be actually accepted.

3.3.2 Method 2: Optimal Flow Placement with Memory

As we mentioned earlier, in the optimization problem formulation of Method 1, we implicitly assume that flow arrivals are independent and "memory-less" (i.e., Poisson): the "average" (or "expected") number of flows arriving in the "next typical time interval" is still λ_i. In Method 2, we will limit the total number of flows of type i in the system (including both the existing ones and those that are to be admitted) to be at most λ_i. Intuitively, we could consider all the existing flows had arrived in the same (typical) time interval as the new flows. In the objective function, we will take the contributions of the existing flows into account as well. (Note this is not necessary, since the contribution from the existing flows is a constant. However, expressing the objective function in this way makes it easier to read.) Let $y_{i,j}$ be the number of the existing flows of type i with bandwidth allocation $c_{i,j}$. The optimization problem formulation is presented below.

3.3.2.1 *Problem Formulation B.2.1: Maximizing Expected Number of Flows*

$$\text{maximize} \quad \sum_{i=1}^{K} \sum_{j \in I_i} [z_{i,j} + y_{i,j}]$$

$$\text{subject to} \quad \sum_{j \in I_i} (z_{i,j} + y_{i,j}) \le \lambda_i, \; i = 1,2,...,K \qquad (1)$$

$$\sum_{i=1}^{K} \sum_{j \in I_i} [z_{i,j} + y_{i,j}] c_{i,j} \le C$$

$$\text{and} \quad z_{i,j} \ge 0, \; i = 1,2,...,K, \text{ and } j \in I_i$$

3.3.2.2 *Problem Formulation B.2.2: Maximizing Expected Utility*

$$\text{maximize} \quad \sum_{i=1}^{K} \sum_{j \in I_i} [z_{i,j} + y_{i,j}] u_{i,j}$$

$$\text{subject to} \quad \sum_{j \in I_i} (z_{i,j} + y_{i,j}) \le \lambda_i, \; i = 1,2,...,K \qquad (2)$$

$$\sum_{i=1}^{K} \sum_{j \in I_i} [z_{i,j} + y_{i,j}] c_{i,j} \le C$$

$$\text{and} \quad z_{i,j} \ge 0, \; i = 1,2,...,K, \text{ and } j \in I_i$$

Note that because of the constraint (1) or (2), $z_{i,j} = 0$ if $\sum_{j \in I_i} y_{i,j} \ge \lambda_i$. Hence the number of flows of type i that are admitted into the system will never exceed the expected number of arrivals. We can replace the constraint (1) in the optimization problem B.2.1 or the constraint (2) in the optimization problem B.2.2 with the following constraint:

$$z_{i,j} + y_{i,j} \le x^*_{i,j}, \; i = 1,2,...,K, \text{ and } j \in I_i \quad (3),$$

where $\{ x^*_{i,j} \}$ are the solutions to the optimization problem A.1 or the optimization problem A.2. Namely, $x^*_{i,j}$ is the (expected) optimal number of flows of type i with bandwidth allocation $c_{i,j}$ that can be admitted into the system (over a typical time interval). Intuitively, whenever the number of flows of type i with bandwidth allocation $c_{i,j}$ that have been admitted into the system exceeds $x^*_{i,j}$, no flows of type i will be considered for a bandwidth allocation of $c_{i,j}$. By using the constraint (3), we, in some sense, attempt to approximate (or converge to) the optimal solutions obtained in the optimization problem A.1 or the optimization problem A.2.

Given the above optimization problem formulation, we have the following probabilistic flow admission control algorithm.

A flow of type i is admitted into the system with probability $p_i = \frac{\sum_{j \in I_i} z_i^*}{\lambda_i}$,

and assigned a bandwidth allocation of $c_{i,j}$ *with probability* $p_{i,j} = \frac{z_{i,j}^*}{z_i^*}$.

3.4 Admission Control for Flows with Finite Duration

In this section, we consider flows with finite duration. When a flow request arrives, it specifies its staring time t_s and t_e. Let $T = t_e - t_s$ be the duration of the flow. To incorporate flow duration into our framework, we assume that the time is slotted into time units (or intervals) of length (the length of a "typical time interval"). Each time unit is considered a "typical time interval", i.e., the expected number of flows of type i arriving during a time unit is λ_i.

Now consider the arrival of a flow of type i whose duration spans T the time units τ_1, τ_2, ... , τ_n. For $k = 1,2,...,n$, let $y_{i,j}^k$ be the number of existing flows of type i during time unit τ_k that have been allocated $c_{i,j}$ amount of bandwidth. Let $C_{res}^k = C - \sum_{i=1}^{K} \sum_{j \in I_i} y_{i,j}^k c_{i,j}$ and $C_{res}^{\min} = \min_{1 \le k \le n} C_{res}^k$. Then using Method 1 in Section 3, where we replace C_{res} with $C_{res}^{\min}$, we can maximize the expected number or expected utility of new flows that can be admitted into the system. Let $\{ z_i^* \}$, $i = 1,2,...,K$ be the solutions to the either of the two optimization problems, where in particular $z_i^* = \sum_{j \in I_i} z_{i,j}^*$, and $z_{i,j}^* \ge 0$ is the expected number of flows of type i with a bandwidth allocation of $c_{i,j}$. Then this new flow of type i will be admitted into the system with probability $p_i = \frac{\sum_{j \in I_i} z_i^*}{\lambda_i}$, and assigned a bandwidth allocation of $c_{i,j}$ with probability $p_{i,j} = \frac{z_{i,j}^*}{z_i^*}$.

Alternatively, we can also formulate the optimization problem using Method 2. In the following, we present the optimization problem formulation for the utility case, where we maximize the *total expected utility* of flows that have been or can be admitted into the system in the duration of the new flow of type i, namely, over the time units τ_1, τ_2,..., τ_n.

3.4.1 Problem Formulation C.2.2: Maximizing Total Expected Utility

$$\text{maximize} \quad \sum_{k=1}^{n} \sum_{i=1}^{K} \sum_{j \in I_i} [z_{i,j} + y_{i,j}^k] u_{i,j}$$

subject to $\sum_{j \in I_i} [z_{i,j} + y_{i,j}^k] \le \lambda_i$, for $1 \le k \le n$ and $i = 1,2,...,K$ (4)

$$\sum_{i=1}^{K} \sum_{j \in I_i} [z_{i,j} + y_{i,j}^k] c_{i,j} \le C, k = 1,2,...,n$$

and $z_{i,j} \ge 0$, $i = 1,2,...,K$, and $j \in I_i$

As before, we can replace the constraint (4) above with the following constraint:

$$z_{i,j} + y_{i,j}^k \le x_{i,j}^*, \quad i = 1,2,...,K, \quad j \in I_i, \text{ and } 1 \le k \le n \quad (5)$$

where $\{ x_{i,j}^* \}$ are the solutions to the optimization problem A.2.

3.5 Flows with Priorities

We now consider the problem of stochastic flow placement and admission control where flows have priorities (or "importance" associated with them).

Suppose there are a total of P priorities, 0, 1,..., P-1, where 0 is the highest priority, and P-1 is the lowest priority. For each flow type i, let λ_i^p be the expected number of flows of type i with priority p, where $p = 0,1,...,P-1$. Let λ_i be the total expected number of flows of type i, i.e., $\lambda_i = \sum_{p=0}^{P-1} \lambda_i^p$.

We can incorporate priorities of flows into our stochastic flow placement and admission control framework by considering flows with highest priority first.

Given the expected number of flows of each type that have the highest priority (i.e., λ_i^0's), we can use the framework discussed in the earlier sections to determine $x_{i,j}^{0,*}$, the expected number of flows of type i that have the highest priority and can be allocated with $c_{i,j}$ amount of bandwidth. Let $C_{res}^1 = C - \sum_{i=1}^{K} \sum_{j \in I_i} x_{i,j}^{0,*} c_{i,j}$. Then C_{res}^1 is the (expected) remaining system capacity that can be used to flows of lower priorities (i.e., priorities 1, ...,P-1).

Note that if $\sum_{j \in I_i} x_{i,j}^{0,*} < \lambda_i^0$ for any i, then we must have C_{res}^1 =0. Hence the (expected) remaining system capacity is zero. In this case, we will consider any flows of lower priorities. If C_{res}^1 is not zero, we will consider flows with priority 1. Again using our above framework (where we replace C with C_{res}^1),

We can determine $x_{i,j}^{1,*}$, the expected number of flows of type i that have priority 1and can be allocated with $c_{i,j}$ amount of bandwidth. Define $C_{res}^2 = C - \sum_{i=1}^{K} \sum_{j \in I_i} x_{i,j}^{1,*} c_{i,j}$. Then C_{res}^2 is the (expected) remaining system capacity that can be used to flows of lower priorities (i.e., priorities 2, …,*P*-1).

As long as C_{res}^2 is not zero, we continue with the next highest priority. The procedure stops whenever the (expected) remaining bandwidth reaches zero.

Using the above procedure, we can determine, $x_{i,j}^{p,*}$ the expected number of flows of type i at a given priority level p and with a given bandwidth allocation $c_{i,j}$. Based on this, we can design a stochastic admission control as before. For each flow type i and priority level p, let $x_i^{p,*} = \sum_{j \in I_i} x_{i,j}^{p,*}$. Then a flow of type i with priority p will be admitted into the system with probability $q_i^p = \frac{x_i^{p,*}}{\sum_{l=0}^{p} \sum_{i=1}^{K} x_i^{l,*}}$, and it is allocated $c_{i,j}$ amount of bandwidth with probability $q_{i,j}^p = \frac{x_{i,j}^{p,*}}{\sum_{l=0}^{p} \sum_{i=1}^{K} x_i^{l,*}}$.

Note that if for some priority level p, $x_i^{p,*}$=0. Then any flows of type i with a priority level p or lower will have a zero probability to be admitted into the system. This may not be desirable if the current system capacity is not zero. We can circumvent this problem by allowing a flow of type i with a priority level p or lower to be admitted into the system as long as $\sum_{l=0}^{p-1} y_i^l < \sum_{l=0}^{p-1} x_i^{l,*}$, where y_i^l is the existing number of flows of type i with priority l, l=0,1,..,p-1.

In other words, flows of type i with higher priorities (0,1,…,p-1) have not fully utilized their expected bandwidth. In this case, a flow of type i with a priority level p or lower can be admitted into the system with an appropriate bandwidth allocation $c_{i,j}$. Later on, when a flow of type type i with a higher priority l, l=0,1,..,p-1 arrives, this flow may be expended from the system, *if necessary*. And the new flow is allocated with the corresponding bandwidth allocation $c_{i,j}$.

3.6 Flow Parameter Estimation

Before we end this report, we briefly discuss how the flow parameters λ_i's can be specified and estimated in practice. To estimate λ_i in a meaningful way, we can either divide the time axis into discrete (i.e., non-overlapping) intervals of length T, or use a smoothing window of length T. Depending on the applications, one of the two methods may be more suitable than the other. In addition, the parameter T should be determined based on the flow arrival rates, desired accuracy of estimation, system performance/complexity, and other factors.

Although throughout the report, we have assumed that λ_i is the *expected* number of flows of type i arriving during a typical time interval T. In reality, it is not necessary to define λ_i to be the average number of flows of type i arriving over intervals of length T. We can define, for example, λ_i to be the expected maximum number of flows of type i arriving during intervals of length. Specifically, let Y_i the random variable denoting the maximum number of flows of type i arriving during an interval of length T. Then we define $\lambda_i = E[Y_i]$.

More generally, we can define λ_i to be any percentile of the flow arrival distribution. Let X_i be the random variable denoting the number of flows of type i arriving during an interval of length T. Let $f(X_i)$ be the distribution of X_i. We can choose λ_i to be such that $\Pr\{X_i > \lambda_i\} \le \varepsilon$, for a given nonnegative number ε, $0 \le \varepsilon \le 1$. Such a general definition of λ_i may be desirable in a system where flows have priorities. For example, we may choose ε^p for each priority p in such a manner that $0 \le \varepsilon^0 \le \varepsilon^1 \le \cdots \le \varepsilon^{P-1}$. In particular, we may choose =0. Hence, λ_i^0 denotes the expected maximum number of flows of type i with highest priority. As a result, our stochastic flow placement and admission control will attempt to accommodate as many flows with the highest priority as can be accommodated by the system capacity.

Section 4 Alternate Optimal Resource Allocation Algorithms

In this section we report on the alternate set of optimal resource allocation algorithms developed by Boston University.

4.1 Background

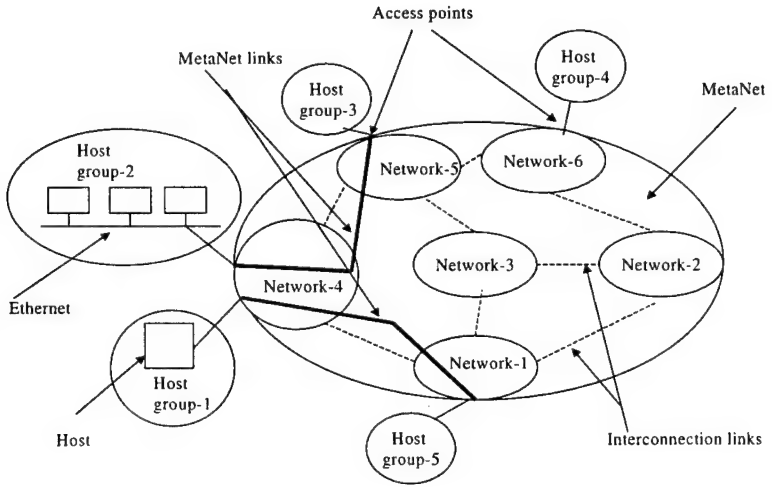


Figure 1. MetaNet, Hostgroups and MetaNet Links

(Courtesy of MetaNet-AIC Interface Functional Specification by Telcordia and JHU, V2, 7/16/99. Slightly modified.)

This work is part of the Coordination and Optimization of Quality-of-Service End-to-end Resources (CONQUER) for Adaptive Information Channels Architecture. The physical environment of the architecture is shown in Figure 1. Hostgroups that consist of groups of user hosts are connected by virtual links denoted by MetaNet link. A host group is connected to the network via an access point that physically may represent a port in an ATM switch, a router or a terminal.

The CONQUER software architecture is shown in Figure 2. The system consists of four major components: IPM (Information Policy Management), AIC (Adaptive information Control), MetaNet, and IDM (Information Dissemination Management). The IPM layer serves as the interface to the commanders in the battlefield. It translates the application requests into resource allocation requests and passes them to the AIC layer. The AIC layer performs two critical functions, policy application and resource management. The MetaNet layer hides the technological details of the underlying network and presents them to the AIC layer as abstract information channels with quality-of-service parameters.

The resource allocation algorithms described in this report are part of the AIC layer. Their primary function is to do the resource reservation and allocation.

4.2 Problem Definition

4.2.1 Input Assumptions

As input to the resource allocation problem, we assume that there is a finite set of channel requests, indexed by $i = 1, ..., N$. Each channel request is characterized by the following attributes:

a. A source identifier, defining the origin group for the data requested by the channel
b. A set of destinations; each destination is viewed as a separate channel request. For the initial algorithms, we assume that the set of destinations contains a single element for each request.
c. A utility function, which defines utility as a function of bandwidth allocated. Specification of the utility function is discussed later.
d. Start time for the channel, quantized to discrete intervals.
e. End time for the channel, quantized to discrete intervals.
f. A request identifier, which we denote by i in the rest of this paper.
g. Priority level, corresponding to a quantized level

For the incremental version of the algorithm, an additional input is the existing state of the network, as defined by the requests that are currently being serviced by the network. Each of these requests is described by the following information:

a. A source identifier, defining the origin group for the data requested by the channel
b. A set of destinations; each destination is viewed as a separate channel request. For the initial algorithms, we assume that the set of destinations contains a single element for each request.
c. A utility function, which defines utility as a function of bandwidth allocated.
d. A bandwidth allocation which is its current quality of service.
e. Start time for the channel, quantized to discrete intervals.
f. End time for the channel, quantized to discrete intervals.
g. A request identifier, i.
h. Priority level.

The resulting resource allocation problems seek to maximize value by allocating quality of service, measured in units of bandwidth, while guaranteeing priority service. We will expand on the objective later.

In addition to information about the requests, the input specifications must also describe the available network. We assume that the network is described in terms of a set of virtual channels, one duplex channel per origin-destination unordered pair, where each channel k has an associated bandwidth capacity C_k. This virtual channel is an abstraction of the physical interconnections provided by MetaNet between origin and destination locations.

At the heart of the optimization function is the utility function associated with each request. This utility function is assumed to be specified by a discrete set of (bandwidth, value) pairs, as illustrated in the figure below.

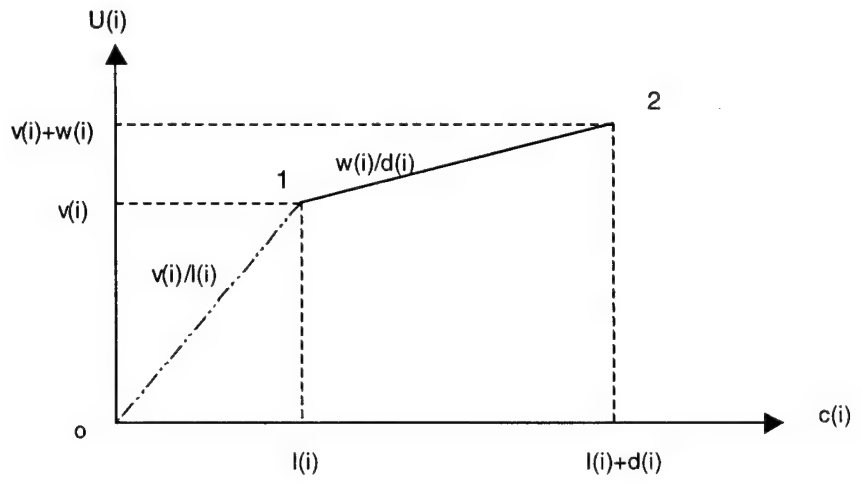


Figure 3. Utility Function of A Request

The pair with the lowest value of bandwidth is assumed to be the minimum connect bandwidth, below which the quality of service is unacceptable. In the diagram above, this is denoted by *I(i),* which achieves value *v(i).* For higher bandwidth allocations, the discrete (bandwidth, value) pairs can be interpolated to describe a piecewise linear function of bandwidth. Adding the default point (0,0), we assume that the interpolated piecewise linear function is concave, and represents the value achieved for any continuous bandwidth allocation greater than the minimum value.

4.2.2 Notation

We use the following notation in the remainder of this report. Let i indicate a report index, and let t determine a time interval index. Then,

a. N is the total number of requests.
b. s_i is the index of the start interval for the request.
c. e_i is the index for the end interval for the request.
d. $U_i(b)$ is the utility of allocating bandwith b to request i.
e. O_i is the origin index of request i
f. D_i is the destination index of request i.
g. b_i is the bandwidth allocated to request i.
h. p_i is the priority of request i.
i. C_{od} is the capacity of the duplex channel, expressed in bandwidth units, interconnecting connecting nodes o and d.
j. L_i is the minimum bandwidth that receives value in the utility function for request i.

k. M_i is the maximum bandwidth that receives value in the utility function for request *i*.
l. *I()* is the indicator function which is 1 if the condition is met, 0 otherwise

With this notation, the decision variables are the allocated bandwidth values b_i for the individual requests. These are constrained by the following relations:

$$\sum_{i=1}^{N} b_i I(\{O_i, D_i\} = \{o, d\}) I(s_i \le t \le e_i) \le C_{od}$$

for all virtual links *{o,d}*and all time intervals *t*, so that the duplex link capacity is not exceeded for any time interval, and

$$L_i \le b_i \le M_i$$

for all requests *i*, so that the bandwidth allocated to each request is within acceptable bounds.

4.2.3 Objective

We assume that priorities are ordered so that more important priorities correspond to lower indices. With this notation, the request acceptance problem can be evaluated hierarchically, priority by priority, as follows. Let J_m denote the value achieved by serving calls at priority *m*, for $m = 1, ..., M$. Given a set of decisions b_i which satisfy the above feasibility constraints, we compute J_m as follows:

$$J_m = \sum_{i=1}^{N} U_i(b_i) I(p_i \le m)$$

We say that a vector $(J_1, ..., J_M)$ achieves better performance than a vector $(J'_1, ..., J'_M)$ if the first vector is greater than or equal to the second vector in a lexicographical order. That is,

- $J_1 > J'_1$, or
- $J_1 = J'_1$, and $J_2 > J'_2$,, or
- $J_1 = J'_1$, and $J_2 = J'_2$, and $J_3 > J'_3$, or ...
-
- $J_1 = J'_1$, and ... and $J_{M-1} = J'_{M-1}$ and $J_M > J'_M$

The above relations merely state the standard rules for priority resource allocation. First, evaluate the resource allocation in terms of how it serves the highest priority. An allocation that achieves more value by serving higher priority requests is preferred. Given two allocations that achieve the same value at the highest priority, the allocation that achieves a better value at the next highest priority is preferred. The recursive preference definition continues until all priorities are defined.

A natural consequence of this objective is that it decomposes the resource allocation problem into a recursive procedure across priorities. First, resources are dedicated to the most important requests; subsequently, resources are assigned to less important requests (in terms of priority) until no further requests can be accommodated. This decomposition is described in the next subsection.

4.2.4 Decomposition by Priorities and Virtual Links

Assume that requests of priority levels $m = 1, ..., k-1$ have already been allocated bandwidth b_i, and that there is still some residual bandwidth remaining during some time intervals. Define the residual bandwidth at priority k as follows:

$$R_{od}(t,k) = C_{od} - \sum_{i=1}^{N} b_i I(\{O_i, D_i\} = \{o,d\}) I(s_i \le t \le e_i) I(p_i < k)$$

The objective for assigning bandwidth to priority k is to maximize the incremental utility $J_k - J_{k-1}$, over the choices of bandwidth assignments b_i to requests of priority k, and over the virtual link $\{o,d\}$, as defined by:

$$J_k - J_{k-1} = \sum_{i=1}^{N} \sum_{links\{o,d\}} U(b_i) I(p_i = k) I(\{O_i, D_i\} = \{o,d\})$$

subject to the feasibility constraints:

$$\sum_{i=1}^{N} b_i I(\{O_i, D_i\} = \{o,d\}) I(s_i \le t \le e_i) I(p_i = k) \le R_{od}(t,k)$$

That is, the requests must satisfy the residual bandwidth constraints for all time intervals on each virtual link. Once priority k is satisfied, the new residual bandwidths $R_{od}(t,k+1)$ are computed, and requests at priority $k+1$ are considered.

The above structure suggests the following decomposition: group all requests by virtual link, and allocate the resources of a virtual link independently, one priority at a time.

Note the following: It is possible for an optimal allocation to contain some requests at priority one which are allocated no bandwidth, whereas some requests at priority 3 or 4 contain bandwidth. This is because the requested start and end intervals for the requests may not be identical for all requests. Thus, if there is an overload of priority 1 requests in a specific time interval, some of these requests will be rejected. Nevertheless, priority 3 requests that do not request this interval may still be assigned bandwidth. Thus, the concept of a "critical level" of priority whereby all lower-indexed priorities are assigned full bandwidth is impossible to define whenever requests have time duration that extends beyond a single interval.

4.2.5 Comments

Our continuous interpolation of the utility function assumes a continuously scaled model of bandwidth. In many applications, a discrete model with specific set points may be preferred. In particular, one can think of multimedia applications using an MPEG coding scheme. If a minimum of 1Kbytes/sec and a maximum of 2 Kbytes/sec are requested, and only 1.5 Kbytes/sec can be allocated, then one out of every four frames can be dropped, or the lowest level of details

in each frame can be omitted during the MPEG encoding process. Extension of our algorithms to discrete models is straightforward.

4.3 Solution Approach

4.3.1 Formulation as Mixed-Integer 0-1 Program

We describe a mixed-integer 0-1 programming formulation to solve the batch problem first. The approach begins by describing a set of tasks to be scheduled, given a residual bandwidth. These tasks are generated by the individual requests at the priority level under consideration, as follows:

Assume that the current priority is k. Thus, we have available residual bandwidth numbers $R_{od}(k,t)$ for every origin-destination set $\{o,d\}$ and every time interval t. For each request i of priority k, we generate a set of tasks T_{ij} where the index j ranges over the set of discrete bandwidth-value pairs described in the utility function. These tasks are described by the following information:

a. $s_{ij} = s_i$ is the index of the start interval for the task.
b. $e_{ij} = e_i$ is the index for the end interval for the task.
c. V_{ij} is the value of the task i.
d. $O_{ij} = O_i$ is the origin index of the task
e. $D_{ij} = D_i$ is the destination index of the task.
f. B_{ij} is the bandwidth requested for the task.
g. F_{ij} is a boolean flag which indicates whether the task is scalable, and thus partial value can be accumulated by assigning a bandwidth less than B_{ij}.

What differs between different values of j for the same request i are the value V_{ij}, the bandwidth requested B_{ij}, and the scalable flag F_{ij}. We describe how to compute these values for a given request below.

Assume that there are J ordered bandwidth-value pairs describing the utility function $U_i(b)$, ordered by increasing bandwidth. A utility function with 4 values is illustrated below.

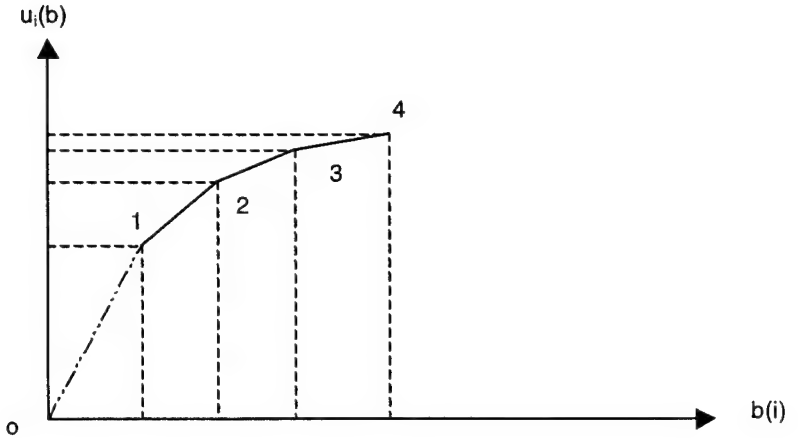


Figure 4. Utility Function of a Request

Denote by (b_{ij},u_{ij}), $j = 1, ..., J$ the ordered pairs in the utility function description for request i. Under the assumption that the resulting interpolated function is concave, we define the following tasks:

- Task T_{i1} has value $V_{i1}=u_{i1}$, with bandwidth $B_{i1} = b_{i1}$, and scalable flag F_{i1} = False.
- Task T_{i2} has value $V_2 = u_{i2} - u_{i1}$, with bandwidth $B_{i2} = b_{i2} - b_{i1}$, and scalable flag F_{i2} = True.
- ...
- Task T_{iJ} has value $V_J = u_{iJ} - u_{i(J-1)}$, with bandwidth $B_{iJ} = b_{iJ} - b_{i(J-1)}$, and scalable flag F_{iJ} = True.

Note that each task denotes an increment in the piecewise linear utility function. The idea is that determining the appropriate bandwidth for a request will be done incrementally, first by assigning bandwidth to the first task (the minimum bandwidth, so it cannot be scaled), then by assigning bandwidth successively to each increment, until no additional bandwidth assignment is possible.

Denote by x_{ij} the fraction of task T_{ij} which is allocated bandwidth. Note that x_{ij} must be in the interval [0,1], and that x_{i1} must be either 0 or 1. Furthermore, one has the constraint that, for $j >$ 1, $x_{ij} \square x_{i1}$, so that no increments can be scheduled unless the minimum bandwidth has already been allocated for that task. The resulting task optimization problem is as follows:

For all tasks defined from requests of priority k, select resource allocation variables x_{ij} to maximize the following objective:

$$\sum_{i=1}^{N} \sum_{j=1}^{J} V_{ij} x_{ij} I\{p_i = k\}$$

subject to the following constraints:

$$\sum_{i=1}^{N}\sum_{j=1}^{J} x_{ij}B_{ij}I(\{O_{ij},D_{ij}\}=\{o,d\})I(s_{ij}\le t\le e_{ij})I(p_i=k)\le R_{od}(t,k)$$

$$x_{i1}\in\{0,1\};\ x_{ij}\in[0,1]\ \text{for}\ j>1;\ \ x_{ij}\le x_{i1}\ \text{for}\ j>1$$

The above formulation describes the batch problem. Next, we focus on describing the setup for the incremental version of the resource allocation algorithm. We make the following assumptions:

- The incremental algorithm is presented with a set of new requests at different priorities.
- The bandwidth allocation of existing requests cannot be modified; the only action on existing requests is to disconnect the existing request

The setup for the incremental problem is identical to the task setup for the batch problem, except as to how existing requests are handled. In essence, each existing request i generates a single task T_{i1} instead of multiple tasks, with parameters $B_{i1} = b_i$, value $V_{i1} = U_i(b_i)$, and scalable flag F_{i1}= True.

Note that it would be easy to relax the above assumption to allow the bandwidth of existing requests to be modified. In essence, the setup for the incremental problem would then be identical to that of the batch problem.

Note also that we have not attempted to account for new arrivals of requests. The implicit assumption is that new requests arrive slower than the planning horizon of existing requests. This initial assumption can be relaxed for future versions of the algorithm.

4.3.2 A Branch-and-Bound Algorithm

Both the batch and the incremental algorithms can be reduced to a unified formulation for each virtual link *{o,d}*, determining which tasks are allocated bandwidth at a given priority level k on that link. To simplify the notation, let w denote a task index for the W tasks considered at priority level k for link *{o,d}*. Then, the problem is to determine the variables x_w given the task descriptions T_w, and the residual bandwidth $R_{od}(t,k)$, and the precedence inequalities among the task variables. The unified formulation can be expressed as:

$$\text{Maximize} \sum_{w=1}^{W} V_w x_w$$

subject to

$$x_w \in [0,1] \text{ if } w \text{ is scalable; } x_w \in \{0,1\} \text{ otherwise} \qquad (3.1)$$

$$x_w \leq x_{pred(w)}$$

$$\sum_{w=1}^{W} x_w B_w I(s_w \leq t \leq e_w) \leq R_{od}(t,k) \text{ for all } t \in \{1,T\}$$

where *pred(w)* is a null index if *w* is a non-scalable task; otherwise, *pred(w)* is the index of the non-scalable task associated with the request which generated task *w*.

We can generate an upper bound on the achievable performance by solving an approximate problem by assuming that each task can be performed in any interval, without regards for continuous allocation of intervals or start times and end times, and that partial allocation of tasks will always result in partial value (all tasks are thus scalable). This latter assumption also removes the need for precedence inequalities, because of the concave nature of the utility function results in optimal allocations which satisfy the precedence inequalities.

The modified upper bound problem can be expressed as a fractional knapsack problem, of the form

$$\text{Maximize} \sum_{w=1}^{W} V_w x_w$$

subject to

$$x_w \in [0,1]; \qquad (3.2)$$

$$\sum_{w=1}^{W} x_w B_w (e_w - s_w + 1) \leq \sum_{t=1}^{T} R_{od}(t,k)$$

The fact that the solution of this problem is an upper bound is easy to verify, as every feasible solution of the original problem is also feasible in this modified problem and has the same value. The solution to this upper bound problem can be obtained as follows. Each task is assigned an index, corresponding to the marginal value per unit bandwidth, computed as

$$Index(w) = \frac{V_w}{B_w(e_w - s_w + 1)} \qquad (3.3)$$

The tasks are ranked by decreasing values of *Index(w)*, and the variables x_w are set to 1 until there is not sufficient bandwidth left to satisfy the bandwidth constraint. The next allocation x_w is then set to the fraction that is required to satisfy the bandwidth constraint with equality. Note that the worst-case complexity of this solution approach is *O(W log W)*, the time required to sort the indices, thus resulting in fast algorithms.

In addition to an upper bound, one can obtain a lower bound on the achievable performance by using a greedy heuristic to solve problem 3.1. The greedy heuristic ranks tasks according to the index (3.3). The algorithm proceeds as follows:

0. Initially, let $R(t) = R_{od}(t,k)$. Form index list of all w, sorted by descending index value.

1. Let w denote the task with largest index remaining

   a. If w is not scalable, check whether $B_w \square\square R(t)$ for all t in $[s_w, e_w]$. If true for all t in the interval, then set $x_w = 1$, remove w from list, and adjust residual bandwidths as $R(t)=R(t) - B_w$ for all t in $[s_w, e_w]$. Continue with step 2.

   b. If w is scalable, check whether $x_{pred(w)}=1$. If not, set $x_w = 0$, remove w from the list and continue to step 2. If $x_{pred(w)}=1$, set

$$x_w = \min\{1, \frac{R(s_w)}{B_w}, \frac{R(s_w+1)}{B_w}, \ldots, \frac{R(e_w)}{B_w}\}$$

   and adjust the residual capacity as $R(t)=R(t) - x_w B_w$ for all t in $[s_w, e_w]$. Remove w from the list, and continue with step 2.

2. If list is not empty, return to 1.

It can be shown that the greedy heuristic above generates a feasible assignment to problem 3.1, and thus produces a lower bound to the optimal value achieved by the solution of problem 3.1. Again, the worst case complexity of the algorithm is *O(WlogW)*.

The above bounds can be used to design a branch and bound algorithm for the exact solution of problem 3.1, which is described in the next section.

4.4 Branch-and-bound Algorithm Design and Implementation

The basic idea in a branch and bound algorithm is to fix the values of some of the integer valued variables (the non-scalable tasks) to either 0 or 1, and to obtain bounds on the achievable performance by computing the upper and lower bound solutions described in the previous section for the remaining variables. The resulting performance can be incorporated into a search strategy.

The 0-1 structure of our discrete decision variables suggests that one can construct a binary tree recursively, by selecting a variable that will be used to form a branch. In the branch-and-bound algorithm, we use the following rule for selecting a branching variable: The last integer variable to be assigned non-zero bandwidth in the upper bound solution is the next variable to be used in the branching value. The reason for this choice is that this integer value is likely to be a "controversial" decision as to whether it is worth allocating bandwidth to this request. Thus, the branch should explore the consequences of making this decision as early as possible.

The branch-and-bound algorithm constructs a tree of nodes composed of decision problems. Each decision problem is characterized by assigning 0-1 values (depending on the branch) to a subset of non-scalable tasks. At each node, upper and lower bounds on the achievable performance are computed as follows: An upper bound is computed by adding the value achieved by the fixed non-scalable tasks to the values computed using the upper and lower bound algorithms described above for the remaining variables.

Given a partial tree of decision problems with associated upper and lower bounds, we proceed as follows:

1. Compute the upper and lower bounds assuming that no non-scalable decision variables are fixed. Use this node as the root node of the tree. Initialize the highest lower bound to the lower bound, and the best lower bound solution to the lower bound solution. Initialize the sorted list of nodes to the root node.

2. Find the most promising leaf node in the sorted list; that is, the leaf node with the highest upper bound. Remove this node from sorted list. If the highest upper bound is less than or equal to the best lower bound found so far, stop and declare the best lower bound solution as the optimal solution. Otherwise, proceed to step 3.

3. Select the branching non-scalable task to branch as the worst-ranked non-scalable task which received bandwidth allocation in the upper bound problem at the leaf. If no such task can be found, remove current node from sorted node list and return to step 1. From the current leaf as parent, form two children leaves in the tree, corresponding to fixing value 0 or value 1 allocation for the branching task.

4. For each of the two child leaves, compute upper and lower bound values as indicated above.

5. Compare the lower bound values for the two leaf nodes to the best lower bound value found for all other nodes. If the best lower bound value for either leaf node is higher, store the highest lower bound leaf value and the associated task allocations as the best lower bound value found so far.

6. Insert each of the two child leaves into sorted list of nodes, and repeat steps 2-6 until an optimal solution is found or no further nodes can be explored.

The above algorithm is guaranteed to stop after enumerating all of the tree nodes corresponding to combinations of non-scalable task values. However, it is not guaranteed to find the optimal solution to problem 3.1, as illustrated by the example below:

Example: Assume that there are three time periods, with available residual bandwidth 1.2, 2.6 and 1.2 respectively. There are three non-scalable tasks of the same priority, described as follows:

- Task 1 has value 6, bandwidth 1, start time 1 and end time 3.
- Task 2 has value 3.8, bandwidth 1, start time 2, end time 3.
- Task 3 has value 3.5, bandwidth 1, start time 1, end time 2.

Note that the tasks are ordered in the same order as the upper and lower bound would consider them. The upper bound computes the total bandwidth of 5 units, and assigns 3 units to task 1 and 2 units to task 2 for a total upper bound value of 9.8. The lower bound starts by scheduling all of task 1 in the intervals 1, 2, 3, thereby leaving residual bandwidths 0.2, 1.6 and 0.2. The next step is to schedule 0.2 of task 2, leaving residual bandwidth 0.2, 1.4, 0. The final step is to schedule 0.2 of task 3, leaving residual bandwidth 0, 1.2, 0, and achieving a total value of $6 + .76 + .7 = 7.46$.

Since there are no non-scalable tasks, the algorithm stops with the assignment of 7.46. Note that this is far from the optimal assignment, which consists of scheduling tasks 2 and 3 in full, plus 0.2 of task 1, for a total value of 8.5, leaving residual bandwidth 0, .4 and 0.

We can easily extend the above branch-and-bound algorithm to an optimal algorithm by using a more sophisticated lower bound whenever there are no non-scalable tasks receiving bandwidth in the upper bound problem. The extension requires computing a tighter upper bound by solving the relaxed linear program associated with problem 3.1, and continuing the logic. If the solution still has no non-scalable tasks receiving bandwidth, then the upper bound is also a lower bound, and an optimal set of bandwidth allocations has been found.

We have chosen not to implement this extension because it is not needed when there is only one request which is scalable. For this special case, the above algorithm is guaranteed to yield an optimal solution. This restricted case was the problem solved in the initial AIM design.

4.5 Algorithm Implementation

The specific implementation of the algorithm is based on the following assumptions:

1. There is a single new request, with a single origin and destination, thus for a single virtual channel.
2. The bandwidth allocation of any existing request on that virtual channel cannot be modified, but that request can be dropped.

Thus, the resource allocation decision is whether to accept the new request on the channel, at what bandwidth, and which of the existing requests should be dropped to make room for the new request.

4.5.1 Input

The following objects, should be input to the optimization subroutine:

a. A table of virtual channel entries, describing the existing requests present in the network, of class AICChannelTable. We assume that the current request is included in this table, and a pointer to this entry is available.

b. A table of available bandwidth for each origin-destination pair per priority, represented as a constraint table. This bandwidth represents the total available bandwidth in each interval for a new request at a specified priority, and thus assumes that lesser priority requests will be preempted.

4.5.2 Output

The output class RAOutput includes the following information: whether the current request is allocated any bandwidth, and, if so, the allocated bandwidth and the list of similar priority existing channels which should be preempted to allocate bandwidth for the current request. It is assumed that all existing channels of lesser priority will be interrupted first before interrupting the indicated channels of the same priority.

4.5.3 Internal Data Structures

The internal data structures required are summarized below:

- **Channel** is the abstraction of the specific virtual channel which is the origin-destination pair for the current request. It is described by a bandwidth parameter.
- **Request** is the abstraction for the tasks described in Sections 3 and 4. It is described in Appendix B.
- **Heap** is the list of active leaf nodes in the branch-and-bound tree, organized in a max-heap indexed by the value of the upper bound so that retrieving the next best upper bound is an O(1) operation. It is described in Appendix B.
- **HeapNode** is the description of a resource allocation subproblem in the branch-and-bound tree. It includes flags designating which variables are still available for optimization, which non-scalable task index is to be used for branching, and the values of the upper and lower bounds.

4.5.4 Algorithm Processing

Step 1. [Initialization] Determine the set of tasks that will be considered in the allocation. The assumption is that all virtual channels are duplex links. Thus, the algorithm first identifies all existing requests in the AICChannelTable which originate at either the origin

node or the destination node of the new request and have the same priority. This is the set of requests that will contend with the new request for bandwidth. It also initializes the available capacity of the channel at each interval to that determined in the constraint table for the priority of the new request.

Step 2. [Task Creation] For the new request, retrieve the discrete points which specify its utility function, and create tasks according to the logic in Sections 3 and 4 for each piecewise linear piece of the utility function. This results in one non-scalable and several scalable tasks for the new request.

For the list of existing requests at the same priority as the new request, and on the same virtual channel, get the current bandwidth allocation and use the request's utility function to compute its utility. Create a non-scalable task for the request with the existing bandwidth allocation and utility, and with start time as the first time interval, end time as the end interval. For times in the task interval, increment the channel capacity by the allocated bandwidth, because this bandwidth is now available for reallocation, in addition to the initial channel bandwidth.

Step 3. [Preprocessing] Compute the value/bandwidth ratio for all the tasks, and sort the tasks in an array in decreasing order of value/bandwidth ratio.

Step 4. [Initialize Branch-and-Bound Heap] Initialize the Branch-and-Bound heap with the task allocation problem with no prior allocation constraints. Compute the upper and lower bounds as described in Section 4 for this problem. Compute the best lower bound solution. Compute also the index of the branching variable in the solution, according to the logic in Section 4.

Step 5. [Recursive Branch-and-Bound Solution]

a. Select problem node at top of heap and remove from heap. If there is no problem, exit.

b. If the upper bound of this problem is within a tolerance value of the best lower bound found so far, exit. The near-optimal solution is found in the best lower bound. Otherwise, continue.

c. If there is no branching variable for this node, return to a and retrieve another problem.

d. Create the left child problem of the current problem node, corresponding to setting the branching variable equal to 0.

e. Compute the upper and lower bounds for the left child problem.

f. If the lower bound is higher than the lower bound of the best solution, make the best solution equal to that of this child problem. Insert the left child into the heap, and reform the heap.

g. Create the right child problem of the current problem node, corresponding to setting the branching variable equal to 1. Check to see whether there is enough available bandwidth in each required interval for the task. If not, return to a. If there is, reduce the bandwidth appropriately in each interval for scheduling the task, mark the variable as assigned, and compute upper bounds and lower bounds for the right child problem.

h. If the lower bound is higher than the lower bound of the best solution, make the best solution equal to that of this child problem. Insert the right child into the heap, and reform the heap.

Step 6. [Output] The best solution found by the branch and bound indicates which tasks were allocated bandwidth, and what fraction of the required bandwidth was allocated. For all tasks generated by the current request, sum up their bandwidths to determine the allocated bandwidth to the current request. For all tasks corresponding to existing requests, if the allocated bandwidth is 0, include the tasks in the preemptable task list.

4.6 Extensions and Suggestions for Future Work

There are three main limitations of the algorithm described above:

1. There is no flexibility in the required start and end times for each request. If each request received a minimum start time and a maximum start time, there would be an additional scheduling problem associated with determining the start time of the request. Extension of the approach described in this report to address this problem in an optimal manner remains an open problem.

2. The abstraction used for the scheduling algorithm assumes that there are no dependencies among virtual channels. This is a major issue with the AIC layer abstract model of the underlying networks. It is difficult to represent explicitly how different origin-destination channels can use the same physical network resources, and thus create interference in the network resource allocation. This problem remains an issue for further investigation.

3. The explicit service model in a time-varying, dynamic environment needs to be modeled better. The current model does not explicitly consider "value lost" by interrupting an existing service on a link. Furthermore, it tries to commit the full set of bandwidths available to serve the current requests. Due to the first year interface restrictions, one was unable to modify the bandwidth allocation to existing requests. Thus, when new requests arrive, one must interrupt existing requests to create room for the new requests. Ideally, we should model the expected arrival of new requests, and reserve some space for important new requests that can interrupt lower-priority requests.

References

[TRW99] TRW S&ITG, "Integration Requirements Document PERFORMANCE ASSESSMENT CAPABILITY FY 1999 (PAC99) Change 1", 10 SEPTEMBER 1999.

[Fal00] B. Falchuk, A. Hafid, Y. Kim, and A. Roy, "Telcordia MetaNet Prototype System and Experiments", Telcordia Deliverable to DARPA, January 2000.

[Funk00] H. Funk, C. Miller, C. Johnson, and J. Richardson, "Applying Intent-Sensitive Policy to Automated Resource Allocation: Command, Communication and Most Importantly, Control. Abstract submitted for presentation at the Fifth Annual Symposium on Human Interaction with Complex Systems", April 30-May 2, 2000, Urbana-Champaign, Ill.

[Mart90] Martello, S. and P. Toth, Knapsack Problems- Algorithms and Computer Implementations, Wiley (1990).

[Lin98] Lin. E., Bibliographical survey of some well-known non-standard knapsack problems, *INFOR Journal*, 36(4), 274-317 (1998).

[Godb99] D. Godbole, V. Gopal, J. Jelinek, B. Morton, D. Musliner, and T. Samad, "Multi Model Predictive Control of Military Air Operations", DARPA-JFACC Symposium on Advances in Enterprise Control, San Diego, November 1999.